Quantum computation with realistic magic-state factories

Joe O'Gorman[1] and Earl T. Campbell[2]
[1]*Department of Materials, University of Oxford, Oxford OX1 3PH, United Kingdom*
[2]*Department of Physics and Astronomy, University of Sheffield, Sheffield S3 7RH, United Kingdom*
(Received 26 August 2016; published 31 March 2017)

Leading approaches to fault-tolerant quantum computation dedicate a significant portion of the hardware to computational factories that churn out high-fidelity ancillas called magic states. Consequently, efficient and realistic factory design is of paramount importance. Here we present the most detailed resource assessment to date of magic-state factories within a surface code quantum computer, along the way introducing a number of techniques. We show that the block codes of Bravyi and Haah [Phys. Rev. A **86**, 052329 (2012)] have been systematically undervalued; we track correlated errors both numerically and analytically, providing fidelity estimates without appeal to the union bound. We also introduce a subsystem code realization of these protocols with constant time and low ancilla cost. Additionally, we confirm that magic-state factories have space-time costs that scale as a constant factor of surface code costs. We find that the magic-state factory required for postclassical factoring can be as small as 6.3 million data qubits, ignoring ancilla qubits, assuming 10^{-4} error gates and the availability of long-range interactions.

DOI: 10.1103/PhysRevA.95.032338

I. INTRODUCTION

Architectures for quantum computers must tolerate experimental faults and imperfections, doing so in the most practical and efficient way. One aspect of fault tolerance is the use of error-correcting codes, which provides a storage method for protecting quantum information from noise. To perform quantum computations, additional techniques are needed to ensure a universal set of quantum gates can be implemented fault tolerantly. Most error-correcting codes natively allow fault-tolerant implementation of gates from the Clifford group, a nonuniversal set of gates. Fully functional quantum computation is attained by adding the Toffoli or $\pi/8$ phase gate to the Clifford group. The prevailing proposal for performing these gates is to first prepare high-fidelity magic states, which are then used to inject a gate into the main computation. These magic states are needed in vast quantities, and their preparation requires a significant portion of a device to operate as a dedicated magic-state factory [1–3]. Alternatives exist to the magic-state paradigm [4–10], but it is unclear whether they will be feasible substitutes due to worse thresholds [11,12].

Magic-state factories use several rounds of distillation protocols, and several directions have been explored [15–20] to improve efficiency over the original proposal which uses Reed-Muller codes [1]. Notably, an $n \to k$ block protocol takes n input magic states and output k at higher fidelity, with higher ratios of n to k generally offering greater efficiency [15–17]. These block protocols do require more complex circuits, but there has been limited investigation into the full resource cost of these protocols. One advance in this direction [3] has shown that block protocols can be realized in constant time, independent of k, by braiding defects in the surface code. Despite this, the same work found that efficiency improvements of block protocols were modest. However, all previous work has taken a very pessimistic estimate of the fidelity of these protocols, leading to an overestimated cost. We present several results improving resource costs and leading to a more optimistic outlook for realzing magic-state factories.

We present a method of realizing the Bravyi-Haah block protocols [16] in constant time without relying on braiding operations, providing a fast implementation to other architectures. Braiding operations natively support control-NOT (CNOT) gates with multiple target qubits in constant time, which is the key step in the circuit reduction of Ref. [3]. Our approach does not require availability of such powerful gates, but rather reduces time costs by borrowing concepts from subsystem codes [21–24] and notions of gauge fixing [7,9,25]. In subsystem codes, operations acting on many qubits are broken up into more pieces each involving fewer qubits, which we find enables greater parallelization in realizing Bravyi-Haah. Our proposal is especially applicable to distributed architectures for quantum computation and provides these devices with an approach to magic-state distillation that is low in ancilla and time costs.

We also show that magic-state factories can make more use of block protocols than previously thought by using techniques to reduce and calculate the global output error. Consider a protocol outputting K qubits with multiqubit global error rate ϵ_g. All previous studies have used the union bound, also called Boole's inequality, to assert $\epsilon_g \leqslant K\epsilon$, where K is the number of magic states and ϵ is the average error rate of single-qubit outputs, obtained by tracing the qubit from the multiqubit state. For an uncorrelated product state, $\rho_g = \rho^{\otimes K}$, we have $\epsilon_g = 1-(1-\epsilon)^K$, so to leading order $\epsilon_g = K\epsilon - O(\epsilon^2)$, and the union bound is a good estimate for small ϵ. However, the output of block protocols can be highly correlated. Boole's inequality holds for correlated states, but ϵ_g can be much smaller than $K\epsilon$. In distillation protocols such as Bravyi-Haah, correlations are produced because block protocols reduce the probability of single errors, but pairs or

clusters of errors can go undetected and lead to a correlated error pair. If one error is present, it almost certainly has a partner. As such, use of the union bound has led to systematic overestimation of noise in Bravyi-Haah and other distillation protocols. Here we track these correlations through multiple rounds of a magic-state factory. We take as our starting point the triorthogonal codes for producing $\pi/8$ magic states [16] and a method of producing Toffoli magic states [26,27]. Once we begin tracking correlations, it becomes apparent that quality control of the factory can be improved. Detecting an error in one part of the factory can, due to correlations, indicate a likely undetected partner error elsewhere in the factory. We introduce the notion of module checking whereby we discard magic states brought into disrepute by their correlated partners. The enhanced fidelity of module checking is both analytically estimated and found by numerical Monte Carlo simulations, with excellent agreement between these methods.

The specification of the distillation protocol is just one aspect of designing magic-state factories. The size of magic-state factories depends on both the distillation protocols and also the underlying error correction codes, and a significant saving can be made by judiciously reducing the error correction code at earlier rounds of magic-state distillation. The balanced investment of qubits at each round of distillation uses small surface codes for low-fidelity magic states and larger surface codes for high-fidelity magic states. A small fraction of magic states in the factory will be high fidelity and require much larger surface codes. Raussendorf *et al.* [28] argued that, consequently, magic-state distillation can be achieved at a constant factor cost over surface code error correction, which we also observe and discuss.

Taking all factors into consideration, we present a blueprint for a factory capable of delivering enough magic states to solve large Shor's algorithm tasks, beyond the reach of classical computation, within a surface code quantum computer. Our results are summarized in Table I, where we look at both the space-time overhead required to produce a Toffoli magic state and the physical footprint of the factory required to produce magic states at an average rate that can just keep up with the "time-optimal" surface code implementation of the algorithm [14], which we further discuss in Sec. VI.

II. REALIZING BLOCK PROTOCOLS

Many descriptions of distillation protocols are high level, leaving open many aspects of how to implement these protocols. To assess the full resource cost, we require a low-level description of distillation protocols in terms of elementary operations, such as one- and two-qubit gates, preparations, and measurements. We call such a description a realization of a protocol, and a given protocol can have different realizations with varying costs.

This section presents a realization of the Bravyi-Haah protocols, giving explicit instructions presented in Fig. 1. The protocol is presented as a four-step process acting on a collection of n noisy $|T\rangle$ states, where $|T\rangle \propto |0\rangle + \exp(i\pi/4)|1\rangle$. The first step uses ancillas to measure operators composed of Pauli-Z operators, and the second step applies a correction dependent on the measurement outcomes. The third step uses ancillas to measure operators composed of Pauli-X operators, with only certain measurement outcomes kept. After a successful third step, the k output magic states are within an encoded state and delocalized across $3k + 8$ sites. The fourth step uses measurements to localize the output qubits to specific sites. All these steps are detailed in Fig. 1 and show how the multiqubit measurements are broken down into ancilla preparation, two-qubit gates, and single-qubit measurements. Observe in Fig. 1 that each multiqubit measurement involves only four entangling gates, with each such gate designated a distinct colored link. Therefore, the measurement is a four-qubit operator. This is called the weight of the operator. When using a single ancilla to measure a stabilizer of weight m, we need m time steps to perform the required controlled-gate operations. The low, and constant, weight of our measurements gives the realization a constant time cost. More commonly, the Bravyi-Haah protocol is presented as requiring only two

TABLE I. The size and time requirements of some examples of magic-state factories. We consider an implementation of Shor's algorithm requiring $40N^3$ Toffoli gates, which dominates the overhead. We realize each of these gates using a single Toffoli magic state or seven T states in parallel [13], whichever proves optimal. In this algorithm, the Toffoli gates are all sequential, so using time-optimal methods [14], the fastest possible run time is $40N^3 t_{\text{meas/ff}}$, where $t_{\text{meas/ff}}$ is the time taken to make a physical measurement and feed forward the result to selectively perform a single-qubit gate elsewhere in the quantum computer. The number of "physical qubits in factory" neglects qubit cost associated with measuring surface code stabilizers, so for many architectures this number will be doubled. The variable t_{sc} is the time taken to perform a single round of the parallel stabilizer measurements of the surface code, a process involving four CNOT gates, two single-qubit gates and a measurement. We assume throughout that $t_{\text{meas/ff}} = 0.1t_{\text{sc}}$, which is reasonable for a distributed architecture such as ion traps.

	Magic states required		Space-time overhead per magic state in qubit rounds		Physical qubits in factory (and evaluation time) required for time-optimal computation			
	Type	Count	$p_g = 10^{-3}$	$p_g = 10^{-4}$	$p_g = 10^{-3}$, $t_{\text{meas/ff}} = 0.1t_{\text{sc}}$		$p_g = 10^{-4}$, $t_{\text{meas/ff}} = 0.1t_{\text{sc}}$	
Problem					$t_{\text{sc}} = 10^{-3}$ s	$t_{\text{sc}} = 10^{-5}$ s	$t_{\text{sc}} = 10^{-3}$ s	$t_{\text{sc}} = 10^{-5}$ s
1000-bit Shor	Toffoli	$10^{10.60}$	1.41×10^7	5.35×10^5	1.73×10^8 **(6.6 weeks)**	1.73×10^8 **(11 h)**	6.30×10^6 **(6.6 weeks)**	6.30×10^6 **(11 h)**
2000-bit Shor	Toffoli	$10^{11.51}$	1.66×10^7	5.71×10^5	2.18×10^8 **(53 weeks)**	2.18×10^8 **(3.7 days)**	6.97×10^6 **(53 weeks)**	6.97×10^6 **(3.7 days)**
4000-bit Shor	Toffoli	$10^{12.41}$	1.94×10^7	6.12×10^5	2.50×10^8 **(8 years)**	2.50×10^8 **(4.2 weeks)**	7.69×10^6 **(8 years)**	7.69×10^6 **(4.2 weeks)**

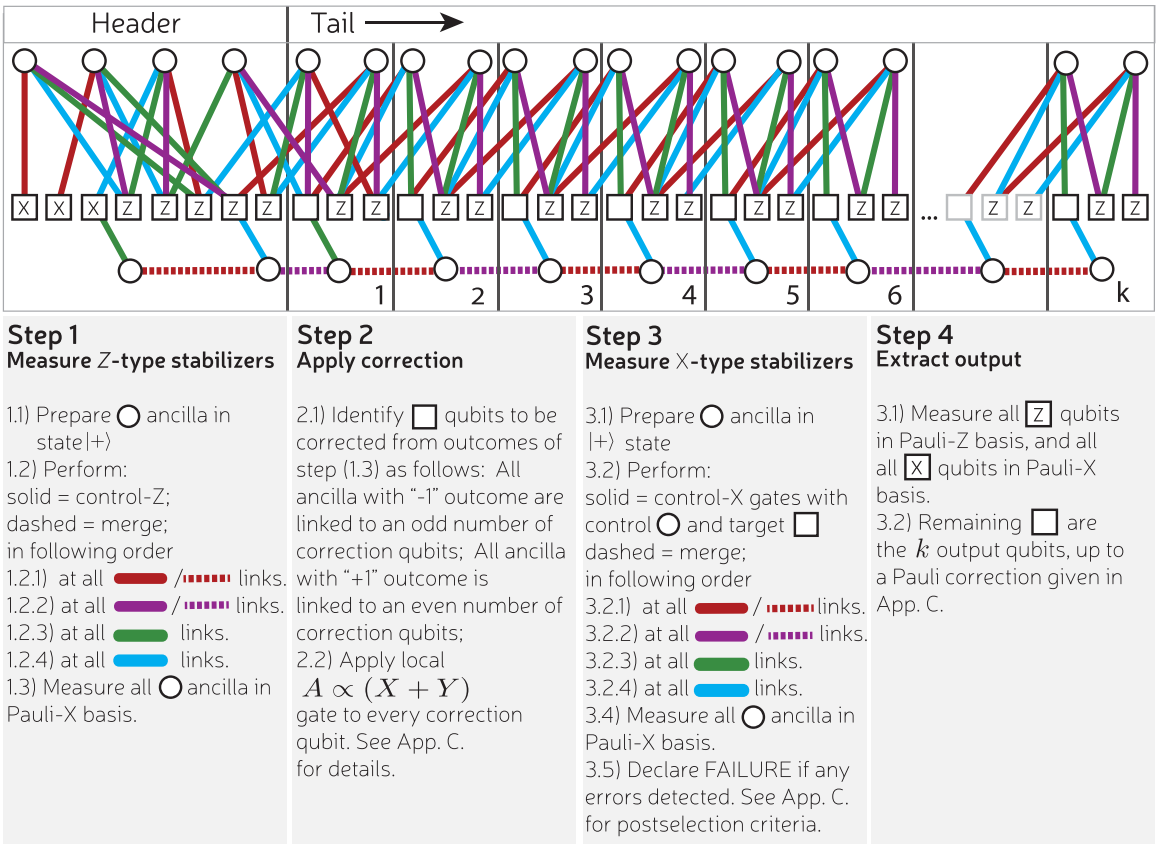

FIG. 1. Explicit circuit for realizing Bravyi-Haah $(3k+8) \to k$ block protocols for $k = 2,6,10,14,\ldots$. Squares indicate the $(3k+8)$ noisy $|T\rangle$ magic states to be distilled, and circles represent ancilla qubits used to effect measurements on the magic states. Increasing k does not increase the number of time steps in the protocol but increases the number of qubits involved in a block. As we increase k, we add qubits to the tail end, with the protocol translationally invariant along the tail. We report that we have independently confirmed the validity of the protocol for $k = 2$ by full wave-function simulation, which further confirmed that all single errors are detected and all two errors processes lead to outputs with correlated errors.

measurements of X-type observables that have weight $2k+4$, implying a potentially expensive time cost. However, the concept of gauge subsystem codes provides a method whereby such complex measurements can be broken down into a larger number of simpler measurements [21–24]. In Appendix C we present a rigorous demonstration that the Bravyi-Haah code can be viewed as a subsystem code, and using a combination of gauge fixing and a cat-state ancilla, we create a circuit of depth 4 for both X and Z measurement sets. We call this general approach gauge-MSD, where MSD is short for magic distillation. We remark that constant time realization was also found by Fowler *et al.* [3] but was tied to a monolithic braiding architecture.

The time complexity of our realization is simply

$$t_{\text{block}} = 8t_{\text{cnot}} + t_{\text{A}} + 2t_{\text{prep}} + 3t_{\text{measure}} \sim 8dt_{\text{sc}}, \qquad (1)$$

where t_{cnot} is the two-qubit gate (e.g., CNOT) time, t_{A} is the gate time for single-qubit $A = (X+Y)/\sqrt{2}$ rotation, t_{prep} is the single-qubit preparation time, and t_{measure} is the single-qubit measurement time. Here all operations are applied fault tolerantly to logical qubits within an error-correcting code. Therefore, the time scales are for fault-tolerant gates. We will assume throughout that logical CNOT gates are applied transversally and thus take time $t_{\text{cnot}} = t_{\text{g}} + d \times t_{\text{sc}}$, where t_g is the time for the physical gate, d is the code distance, and t_{sc} is the time for a round of stabilizer measurements. Therefore, for large distances the time is dominated by $8dt_{\text{sc}}$. We will also take this as the time cost of a merge operation [29,30] which we use to project two ancillary qubits into the even- or odd-parity subspace. We assume entangling gates can be performed in parallel, but a qubit can participate in only one gate at a time. We allow entangling gates to be long range as is feasible within distributed architectures for quantum computing [31–38], although this may be relaxed at only a modest increase in resources.

Our realization uses a number of ancilla qubits, so that in addition to the $n = 8+3k$ qubits being distilled we also use $n_{\text{anc}} = 3k+6$ ancilla qubits, giving $n_{\text{tot}} = 6k+14$ logical qubits in total. These ancillas appear as circles in Fig. 1. With these logical qubits encoded in a distance d toric code, the total qubit cost is $N_{\text{tot}} = (6k+14)d^2$. Therefore, the total space-time cost amounts to $N_{\text{tot}}t_{\text{block}} \sim (48k+112)d^3$, which compares well against other approaches (see Appendix A).

When using the toric code, A gates are also not naturally fault tolerant and thus require their own process of state distillation and injection. The time t_A to implement such a gate is therefore the time taken by the logical CNOT which teleports the gate into the computation plus any Clifford correction that

is required, although this can be rolled into a later Clifford operation. The resources required for the state distillation of A are far less than those of the T gate, and the ancilla resource is reusable for many A gates [17,39]. As such, we neglect the overhead of these gates as a small additional overhead to the main process of $|T\rangle$ distillation.

III. OVERVIEW

A. Blocks, branches, and modules

We begin by introducing some helpful vocabulary for describing magic-state factories. Efficient distillation uses $n \to k$ block protocols that take n noisy $|T\rangle \propto |0\rangle + \exp(i\pi/4)|1\rangle$ and with some probability output k states of a higher fidelity. Such a process we call a block, and the previous section described the details of the inner working of such a block. Now we treat each block as a black box with known relations between input and output and consider how these blocks are composed together.

Distillation protocols have many levels forming a treelike structure with many branches that merge at points we call modules, shown in Fig. 2. Branches contain many qubits, which are potentially correlated. However, the inputs to block protocols must not be correlated, so each qubit in a branch must be fed into a different block. Therefore, as we enter a module, a branch of B_l qubits is split up so that each qubit enters a different block. If each block implements a $n_l \to k_l$ protocol, then the whole module can be thought of taking $B_l n_l$ inputs to $B_l k_l$ outputs. This entails that $B_{l+1} = B_l k_l$ and that each module has n_l branches feeding into it. Initially, branches are single qubits, $B_1 = 1$, and so $B_l = \prod_{1 \leqslant j < l} k_j$.

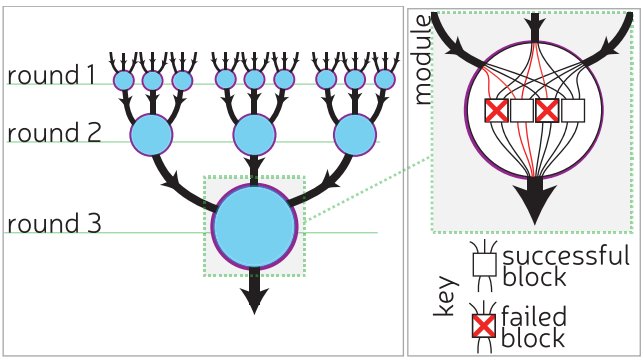

FIG. 2. The tree structure of many rounds of distillation, with branches (directed black lines) that merge at branching points that we call modules. The thickness of the branch increases with each round. The main plot shows a fictitious scheme where $n_l = 3$ and $k_l = 2$ for all rounds. Inset: The structure within a module. Incoming branches contain many qubits; here this is shown to be four. These qubits undergo a permutation σ and are fed into an instance of a block of a distillation protocol shown as a square. Here the three incoming branches carry four qubits, and so we need four instances of a $3 \to 2$ protocol. We use a fictitious protocol to keep the numbers low enough to illustrate clearly. Each of the four blocks output two qubits, and these are merged into a branch of eight qubits fed into a later module. A pernicious error pattern is shown in red (lighter gray), which is detected in two of distillation blocks, marked with crosses, but goes undetected in a third block.

This module-branch structure is common to all proposals to date. Such explicit terminology has not previously been introduced but rather has been left as an implicit consequence of statements about correlation avoidance. Establishing clear vocabulary about this structure is important as we delve into the effect of postselecting at different levels on this structure. Previous protocols have considered whether individual blocks succeed or fail; we call this *block checking*. Below, we outline why it can be advantageous to postselect on the level of the modules, which are collections of blocks. We propose an additional quality check, so that the whole module is discarded whenever any of its blocks fail. We call this *module checking*.

A block will always detect a single incoming error but might fail to detect a pair of errors. When a block detects an error, it indicates the presence of damaged branches, and since errors cluster together within branches, this increases the likelihood of errors in other blocks throughout the module. Consider when two branches fail, each with a correlated error pair; the first branch sends damaged qubits to blocks 1 and 2, and the second branch sends damaged qubits to blocks 2 and 3. Since blocks 1 and 3 each received a single erroneous qubit, they will detect them. But block 2 receives a pair of errors, so they may go undetected. A simplified illustration of this process is shown in Fig. 2. Module checks improve fidelity by preventing these processes from degrading the output fidelity. Even with module checks, it is possible for a pair of corrupted branches to go undetected, but both branches must carry exactly the same pattern of errors, which is a very rare occurrence.

IV. THE G-MATRIX FORMALISM

Bravyi and Haah introduced a matrix description of their $n \to k$ block protocols for $|T_0\rangle$ state distillation. The so-called G matrix is split into two submatrices, G_1 and G_0, with G_0 describing the postselection criteria and G_1 accounting for how input qubits are related to output qubits. For distillation of $|T_0\rangle$ magic states, the G matrix must have the property of triorthogonality that the reader can learn about in Ref [16]. Rather, here we give a pragmatic account of the block's performance. We use $|T_0\rangle \propto |0\rangle + e^{i\pi/4}|1\rangle$ for a magic state and $|T_1\rangle = Z|T_0\rangle$ for the orthogonal state with a Z error. A pure multiqubit state $|T_{x_1}\rangle|T_{x_2}\rangle \cdots |T_{x_n}\rangle$ we concisely represent with the vector $x = \{x_1, x_2, \ldots, x_n\}$. If we apply a block protocol to state x, the block succeeds (detecting no errors) if $G_0 x = 0 \pmod 2$, where the math is performed modulo 2. When successful, the block outputs a state $y = G_1 x$. Noisy magic states will be some probabilistic ensemble over x, with probability $\Pr(x)$. The protocol will detect no errors and output state $|T_{y_1}\rangle|T_{y_2}\rangle \cdots |T_{y_k}\rangle$ with (unnormalized) probability

$$\Pr_{\text{unnorm}}(y) = \sum_{\{x: G_0 x = 0, G_1 x = y\}} \Pr(x). \tag{2}$$

The total success probability is captured by the sum over all possible output states, so

$$\begin{aligned} P_{\text{suc}} &= \sum_y \Pr_{\text{unnorm}}(y) \\ &= \sum_y \sum_{\{x: G_0 x = 0, G_1 x = y\}} \Pr(x) = \sum_{\{x: G_0 x = 0\}} \Pr(x). \end{aligned} \tag{3}$$

Conditioned on success, the normalized distribution on output states is $\Pr_{\text{out}}(y) = \Pr_{\text{unnorm}}(y)/P_{\text{suc}}$. Given an explicit form for G_0 and G_1, this completes the black box picture of block protocol performance. These formulas form the basis upon which we build both our analytic and numerical analysis in the following sections.

The G-matrix formalism of Bravyi and Haah has been significantly generalized [40,41]. This extension provides protocols that convert noisy T magic states into another species capable of injecting complex multiqubit circuits. Included in this framework are protocols, based on G matrices, which provide resources for implementing Toffoli gates. Protocols independently proposed by Jones [27] and Eastin [26] realized error-suppressed Toffoli gates, and here we consider a variant based on G matrices that we discuss further in Appendix E. All three variants perform identically when we use block checking. However, with the G-matrix formalism we can again use module checking to track correlations and achieve superior error suppression. This is just one additional application of module checking; the technique can be deployed in conjunction with the general class of protocols introduced in Refs. [40,41].

V. ANALYSIS OF MODULE CHECKING

We present a method of tracking the leading-order errors, accounting for correlations, through many rounds of module-checked protocols. At each level of distillation the protocol is characterized by a function η that summarizes how well it tolerates leading-order errors.

Definition 1. For every distance-2 G-matrix code that distills $n \to k$ qubits, we define a function $\eta : \mathbb{Z}_2^k \to \mathbb{Z}$, taking values

$$\eta(y) := \#\{y : |x| = 2, G_0 x = 0, y = G_1 x\}, \tag{4}$$

where $|\cdots|$ is the weight $|y| = \sum_j y_j$ and # counts the number of elements in a set $\{\cdots\}$. In other words, the value $\eta(y)$ counts the number of inputs x such that (1) they are weight 2 [formally, $(|x| = 2)$], (2) they are undetected by the protocol (formally, $G_0 x = 0$), and (3) they give y as output (formally, $G_1 x = y$).

Since $\eta(y)$ counts the number of lowest-weight errors leading output y, the total error rate for one round of distillation can be simply estimated as

$$\epsilon_g = \left(\sum_y \eta(y)\right)\epsilon^2 + O(\epsilon^4). \tag{5}$$

Counting errors over many rounds is a more subtle problem, but we find that η still provides sufficient information to perform this calculation. If each round can even use a different protocol, we label the corresponding function with a subscript. We now state our key result.

Theorem 1. Consider L rounds of distillation with module checking, with associated functions $\eta_1, \eta_2, \ldots \eta_L$. Such a protocol outputs a multiqubit magic state where the lth-level modules succeed with probability

$$P_{\text{suc},l} \simeq \frac{A_l + B_l}{(A_{l-1} + B_{l-1})^{n_l}} \tag{6}$$

and output states have global infidelity

$$\epsilon_g^{(l)} \simeq \frac{B_l}{A_l + B_l}, \tag{7}$$

where we have made use of the following:

$$A_l = (1-\epsilon)^{(n_1 n_2 \cdots n_l)}, \tag{8}$$

$$B_l = C_l \epsilon^{2^l} (1-\epsilon)^{(n_1 n_2 \cdots n_l - 2^l)}, \tag{9}$$

$$C_l = \prod_{j=1}^{l} \left(\sum_v \eta_j(v)^{2^{l-j}}\right). \tag{10}$$

The most important quantity in Theorem 1 is C_l. After two rounds C_2 is simply $[\sum_y \eta_1(y)^2][\sum_y \eta_2(y)]$. After l rounds, C_l is a product of l terms. One may approximate the theorem to leading order $\epsilon_g^{(l)} \sim C_l \epsilon^{2^l}$. However, our later numerical investigations found that such an approximation was too coarse, and we really need the slightly more complex form given in the theorem. We postpone proof of this result to Appendix F.

For the Bravyi-Haah protocols it is easy to verify the following:

$$\eta_{\text{BH}_k}(y) = \begin{cases} 3, & |y| = 2, \\ 4, & |y| = k, \\ 0, & \text{otherwise}, \end{cases} \tag{11}$$

so that for $k > 2$ we have

$$\sum_y \eta_{\text{BH}_k}(y)^m = 4^m + 3^m \binom{k}{2} = 4^m + 3^m \frac{k(k-1)}{2}, \tag{12}$$

where the binomial factor arises in counting the number of y where $|y| = 2$. When $k = 2$, we have a special case as then the $|y| = 2$ terms and $|y| = k$ terms are the all same, so

$$\sum_y \eta_{\text{BH}_2}(y)^m = 7^m. \tag{13}$$

For the Toffoli protocol we have

$$\eta_{\text{Tof}}(y) = 4, \quad y \neq 0, \tag{14}$$

so

$$\sum_y \eta_{\text{Tof}}(y)^m = 7 \times 4^m. \tag{15}$$

Given expressions for $\eta(y)^m$, we can compose these protocols anyway we wish and obtain an estimate of the global error rate as given in Theorem 1.

We perform numerical Monte Carlo simulations by sampling from the probability distribution of the raw magic states, where $\Pr(x) = \epsilon^{|x|}(1-\epsilon)^{N-|x|}$, and track its evolution through the magic-state factory.

Our simulations track the progress of potentially damaging input error configurations of the initial raw magic states though the factory. Direct Monte Carlo simulation is possible for one to two rounds of distillation, but with three or more rounds the failure events become so rare that brute force simulation fails to provide statistically meaningful data. Rather, we use a "rare-event" technique that preselects cases with two or more corrupt branches. A full description of the method employed can be found in Appendix G. We thus investigate the performance of a factory which makes use of module checking for a range of raw

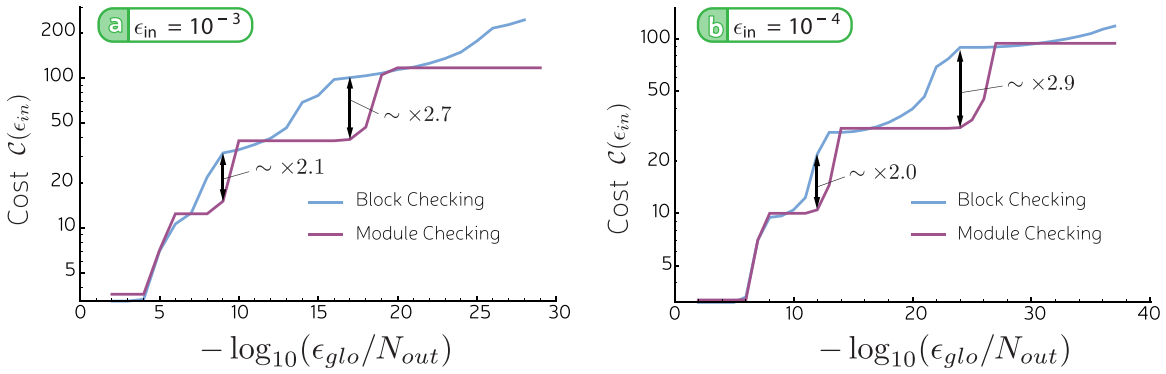

FIG. 3. (a) and (b) The cost $\mathcal{C}$ of the Bravyi-Haah protocols utilizing both block checking and module checking. It can be seen that only around the transitions to an additional level of distillation is block checking very slightly preferable.

magic-state error rates between 0.1% and 1%. We find that the leading-order analytic estimates match well with the numeric results, with the difference between the two being of the order of a few percent in the investigated parameter regime. In fact, wherever the block protocols are involved, if $k \leqslant 14$, we find the percentage difference between the numerical simulation and analytic estimate is $<10\%$. This discrepancy between the analytic estimate and numerical simulations is not visible on log-log plots presented in Appendix G.

The cost of a protocol is the average number of raw magic states consumed to produce one higher-fidelity magic state. For an $n \to k$ protocol, which takes in states with error p, this is

$$\mathcal{C}(p) = \frac{n}{kP_s(p)}. \tag{16}$$

For l rounds of distillation we have $\mathcal{C}_l(p) = \prod_{i=1}^{l-1} \mathcal{C}_i(p_i)$.

Figure 3 compares the cost of a Bravyi-Haah magic-state factory with block checking and module checking, showing the minimum cost achievable for given output error. For output error rate and success probability we use known expressions for block checking, and for module checking we use the success probability and global error given by Theorem 1 with an estimate of the reduced error rate on a single qubit given by the global error rate estimate divided by the total number of qubits in this factory's output.

We find that module checking is superior to block checking for a large proportion of target error rates and can use up to 4 times fewer raw magic states in some regimes. However, near a transition from j to $j+1$ rounds of distillation, module checking loses it advantage and may even be slightly outperformed. The best error rates that can be achieved for a given number of rounds use low-k block codes; for these the benefit in the global error rate of module checking over block checking is smaller (see Table II), while the success probability is, of course, much inferior. Above the transition higher k values are used, for which the success probability is much lower, and this washes out the benefit of the superior error suppression over block checking, which has a higher success probability. In these regimes, as one might expect, module checking is not the optimal approach. In those regions between the transitions, module checking allows the use of higher-k protocols, which are more efficient, to achieve the error rates of lower-k block-checked protocols.

VI. FACTORY OVERHEAD ANALYSIS

While the "cost" is a useful guide to the performance of a distillation protocol, it fails to capture several important features of real magic-state factories. The very purpose of magic-state distillation is to supplement the shortcomings of a particular error-correcting code, which is to say that a magic-state factory is implemented at the level of logical qubits, with the quantum information already encoded. As such, the more relevant question is not how many noisy input magic states are required per output state, but rather the number of the physical qubits that will build up such a factory and the rate at which the distilled magic states are produced. Both of these numbers are of key importance to determining the size of a quantum computer and the run time of an algorithm. A single number, the "space-time overhead" of the magic-state factory, captures these both as a figure of merit, which was also studied in Ref. [3]. In this section we present a comparison based on the space-time overhead of implementing a magic-state factory in a surface code quantum computer, utilizing module checking, where appropriate, and our implementations of the Reed-Muller, Bravyi-Haah, and Toffoli protocols.

We consider a number of issues in this estimate of the footprint of a magic-state factory, including (1) *balanced investment*, the use of smaller surface codes during early rounds of magic-state distillation, and (2) *clock-rate zoning*, cycling through distillation attempts faster during early rounds of magic-state distillation. We will assume throughout that we use the method outlined by Li [42] to inject the initial raw magic state into a logical surface code. We will therefore assume throughout that the initial error rate on a magic state before distillation is $\epsilon_{\text{in}} = 0.4p_{\text{g}}$. We also use the "rotated-lattice" surface codes [29], such that a distance d surface requires d^2 physical data qubits. Of course a practical surface code requires the use of physical ancilla qubits to make the stabilizer measurements of the code; we leave this as an extra multiplicative factor to be applied to our overhead calculated here, as different physical realizations have different requirements. We estimate the surface code distance required to protect a logical qubit up to error rate P_l using the relation $P_l(d, p_{\text{g}}) = d(100p_{\text{g}})^{\frac{d+1}{2}}$ [3].

Given an algorithm and some implementation of it, the number of magic states required is determined in advance. For a given distillation protocol we must then determine whether

TABLE II. The leading coefficient C_l for a variety of protocols with two and three levels of distillation. For clarity we also show C_l in the large block limit ($k \to \infty$). When we write BH_k, we implicitly assume $k > 2$, as the results differ slightly for the $k = 2$ case. The final column shows the ratio between the union bound estimate made by utilizing the reduced error rate on a single qubit ϵ_{BH} made by Bravyi and Haah and the corresponding estimate of the global error rate given by $C_l\epsilon^{2^l}$. It can be seen that the benefit (in error rate) of module checking scales with both k and the number of rounds of distillation.

Level 1	Level 2		C_l	$\lim_{k\to\infty} C_l$	$\lim_{k\to\infty} k_1k_2\epsilon_{\mathrm{BH}}/\epsilon^4$
BH_{k_1}	BH_{k_2}		$\left[16+\frac{9}{2}k_1(k_1-1)\right]\left[4+\frac{3}{2}k_2(k_2-1)\right]$	$\frac{27}{4}k_1^2k_2^2$	$27k_1^3k_2^2$
Tof	BH_k		$112\left[4+\frac{3}{2}k(k-1)\right]$	$168k^2$	$2352k^2$
BH_k	Tof		$\left[16+\frac{9}{2}k(k-1)\right]28$	$126k^2$	$252k^3$
Level 1	Level 2	Level 3	C_l	$\lim_{k\to\infty} C_l$	$\lim_{k\to\infty} k_1k_2k_3\epsilon_{\mathrm{BH}}/\epsilon^8$
BH_{k_1}	BH_{k_2}	BH_{k_3}	$\left[256+\frac{81}{2}k_1(k_1-1)\right]\left[16+\frac{9}{2}k_2(k_2-1)\right]\left[4+\frac{3}{2}k_3(k_3-1)\right]$	$273.375k_1^2k_2^2k_3^2$	$2187k_1^5k_2^3k_3^2$
Tof	BH_{k_1}	BH_{k_2}	$1792\left[16+\frac{9}{2}k_1(k_1-1)\right]\left[4+\frac{3}{2}k_2(k_2-1)\right]$	$12096k_1^2k_2^2$	$28^4\times 3^3k_1^3k_2^2$
BH_{k_1}	BH_{k_2}	Tof	$\left[256+\frac{81}{2}k_1(k_1-1)\right]\left[16+\frac{9}{2}k_2(k_2-1)\right]28$	$5013k_1^2k_2^2$	$20412k_1^5k_2^3$

its magic-state factory is capable of producing magic states of fidelity great enough that there is a high probability that none of the magic states required for the algorithm fail. Thus, we set a target global error rate that our factory must achieve:

$$\epsilon_{\mathrm{target}} = 1 - (P_{\mathrm{suc,alg}})^{1/N}, \tag{17}$$

where $P_{\mathrm{suc,alg}}$ is the desired success probability of our algorithm (i.e., the probability that every non-Clifford gate works) and N is the number of successful iterations of the factory needed to produce the desired number of magic states. For example, a magic-state factory which utilizes three rounds of Bravyi-Haah distillation with a k value of 10 in each round will produce 10^{15} $|T\rangle$ states after 10^{12} successful iterations. A 90% success probability for the algorithm as a whole then implies $\epsilon_{\mathrm{target}} = 1.05 \times 10^{-13}$. We must then check that the factory is capable of producing an output of this quality. If the factory is module checked, then this "10-10-10" factory has a global error rate $\epsilon_{\mathrm{glo}} = 2.3 \times 10^{-16}$, making this a valid protocol. However, the estimate of the global error rate of a block-checked factory gives $10^3 \times \epsilon_{BC} \sim 10^{-11}$, so this would not be a valid factory for this task.

A. Balanced investment

Once we have established that a factory is valid, we can calculate the space-time overhead per magic state that it produces. This is done by calculating the distance of surface code required at each level of distillation d_i, the length of time in surface code cycles that each round of distillation will take T_i, and the number of logical qubits Q_i, including logical ancillas, required at each level of the factory.

Determining the distance required at each level requires slightly different methods depending on whether module or block checking is used. To obtain the benefits of module checking we cannot make full use of balanced investment at the lower levels because doing so would inject noise at a rate comparable to or greater than the rate of correlated error. We can estimate the total global error output in the output of a $n \to k$ protocol as $\epsilon_{\mathrm{tot}} \sim \epsilon_{\mathrm{glo}} + k\epsilon_{\mathrm{enc}}$, where ϵ_{enc} is the random error rate resulting from the size of the logical encoding chosen. As such we determine the distance of the code at the intermediate levels based on a desired error rate ϵ_{enc} to be $0.1\epsilon_{\mathrm{glo}}/k$ to ensure that the error due to each qubit's finite encoding is much less than the reduced error $\sim\epsilon_{\mathrm{glo}}/k$ on that qubit. The final output of the factory can then ne encoded according to $\epsilon_{\mathrm{target}}$.

For a block-checked factory (or the original 15-to-1 Reed-Muller protocol) we do not have worry about "protecting" correlated errors. This means we can work backwards from our error target to determine an efficient balanced investment of qubits, as illustrated in Fig. 4. For a local target error of $p_{\mathrm{top}} = 10^{-14}$ the top level of distillation needs $vP_L(d,\epsilon_{\mathrm{enc}}) < 0.1 \times 10^{-14}$, where v is the space-time volume of a single block of distillation, which is the number of surface code qubits in the block multiplied by the number of times they undergo d rounds of surface code stabilizer measurements. Again, we use a distance corresponding to a lower error rate by a factor of 10 to suppress any error injected by the logical circuitry above that which is left over after distillation. The distance required for the next level down can then be determined by, in this example, $p_{i-1} = \sqrt[3]{p_{\mathrm{top}}/35}$ for the 15-to-1 protocol, which gives distance d_{i-1}, and so on until $p_i > p_{\mathrm{in}}$.

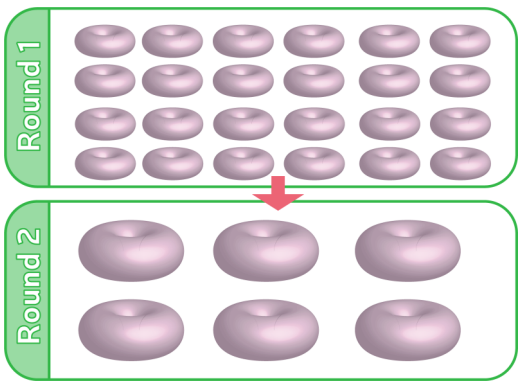

FIG. 4. The concept of balanced investment. During the initial rounds of distillation smaller surface codes can be used to encode the logical information. The factory requires only logical surfaces of the distance corresponding to the target error rate of the round in which that qubit is involved. This target in each round will depend on the final error target, the protocol being used to achieve it, and the error rate of the raw state.

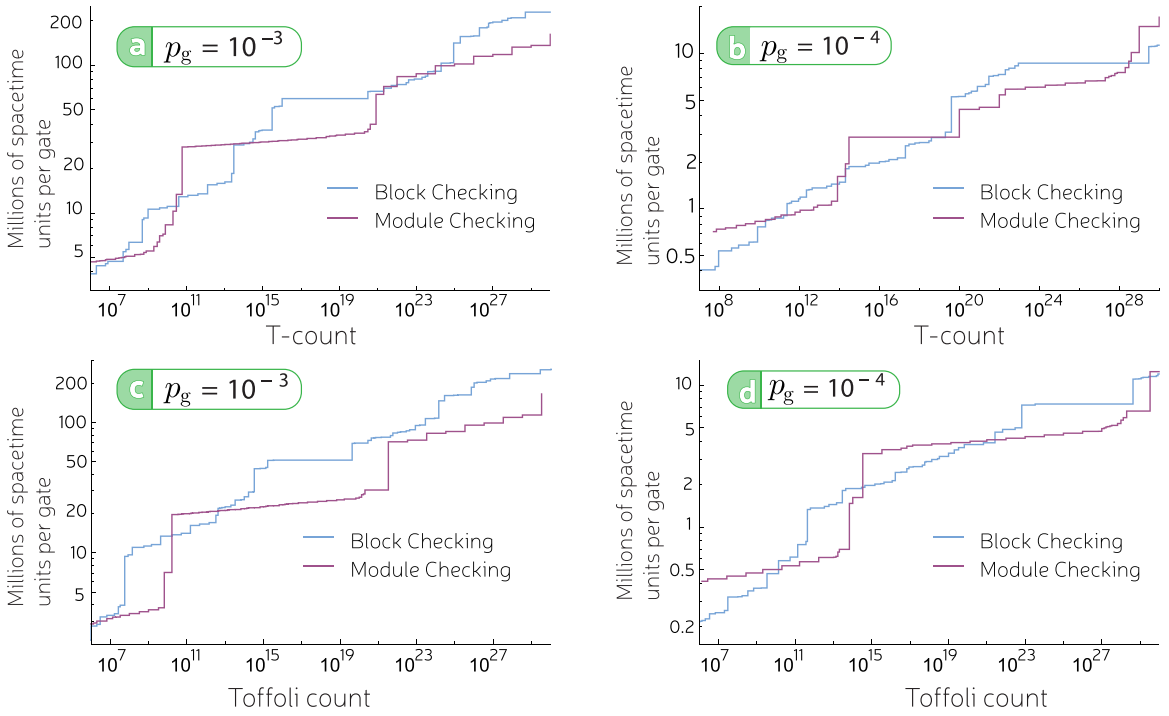

FIG. 5. Space-time overheads of producing both (a) and (b) $|T\rangle$ states and (c) and (d) $|\text{Toff}\rangle$ states. Note that module checking is more beneficial when one of the rounds of distillation is Toffoli.

The length of time T_i for each round of distillation can be simply determined by the protocol used and how many times we attempt it before abandoning the round. We assume that measurements can be completed in one time step, and the time scale of the CNOT gates, preparation, and A gates is dominated by the requirement for d rounds of stabilization afterwards. Therefore, we let the time for each of these be $d \times t_{sc}$. The time taken to implement the distillation protocol in round i is then

$$\tau_i = \begin{cases} 11t_{\text{sc}} \times d, & \text{round } i \text{ uses Bravyi-Haah,} \\ 12t_{\text{sc}} \times d, & \text{round } i \text{ uses Toffoli,} \\ 13t_{\text{sc}} \times d, & \text{round } i \text{ uses Reed-Muller.} \end{cases} \tag{18}$$

B. Clock-rate zoning

Balanced investment also allows another advantage. In the context of surface code computing, the distance of the code is relevant for not only the spatial dimensions of the computer but also the execution time. A surface code of distance d must undergo d rounds of parallel stabilizer measurements to protect from measurement errors. As such, the time taken for a logical operation is proportional to d (except those which can be handled in software), and therefore, using balanced investment, the initial rounds of distillation will take less time. In the hypothetical case that a round of distillation leads to a squaring of the input error rate, this would correspond to a doubling of the code distance required by the next round (by the exponential suppression of error with distance of a subthreshold surface code). Therefore, one can repeat the first round of distillation twice in the time taken for the second round of distillation to be completed. This increases the chance that all the necessary magic states for the second round will have been produced in time for the next round of distillation, without the need to decrease the rate of the factory. As such, the time taken for a round to complete in the distillation factory is $T_i = \tau_i t_i$, where t_i is the number of attempts that you allow at each round. In all our simulations, any "idle time" that the qubits experience is counted towards the space-time cost.

C. Numerical simulations

We have therefore arrived at the expression for the full space-time volume $\mathcal{V}$ in qubit rounds occupied by a factory,

$$\mathcal{V}(\epsilon_{\text{in}}, p_{\text{g}}, \mathcal{N}, \{k_i\}, \{t_i\}, P_{\text{suc,alg}}) = \frac{\sum_{i=1}^{r} Q_i T_i d_i^2}{\prod_i^r k_i \mathcal{P}_{\text{suc},i}} = \frac{\sum_{i=1}^{r} Q_i \tau_i t_i d_i^3}{\prod_i^r k_i \mathcal{P}_{\text{suc},i}}, \tag{19}$$

where k_i labels the magic-state protocol used at each round and $\mathcal{P}_i$ is the probability that round i of distillation will succeed in producing enough magic states to feed the next round. All rounds must succeed for the factory to successfully output $\prod_i^r k_i$ magic states. As discussed, $\mathcal{V}$ is a function of several variables, the raw magic-state error rate ϵ_{in}, the error of gate operations p_{g}, the total states required $\mathcal{N}$, the protocol chosen $\{k_i\}$, the repetitions allowed in each round $\{t_i\}$, and the probability with which you wish the algorithm to succeed $P_{\text{suc,alg}}$.

In practice we calculate $\mathcal{V}$ for one, two, and three rounds of distillation using combinations of the Bravyi-Haah, Reed-Muller, and Toffoli protocols with our proposed implementations, with both block and module checking, and a variety of combinations of t_i. We then search for the most space-time efficient method of producing either T or Toffoli magic states, given a p_{g}, assuming $\epsilon_{\text{in}} = 0.4p_{\text{g}}$, the number of magic states desired, and an arbitrary choice of 90% for $P_{\text{suc,alg}}$.

Figure 5 shows the results of these simulations which suggest that module checking can provide an improvement of a factor of 3 in the space-time overhead in certain parameter regimes. We also see that in some regions module checking can be detrimental by a small amount, such as near the transition from i to $i+1$ rounds of Bravyi-Haah.

We use the numbers we have generated to estimate the size of a magic-state factory required to perform some postclassical factoring tasks using Shor's algorithm. Our results are summarized in Table II, in which we approximate Shor's algorithm as modular exponentiation, as this is by far its most expensive part, and choose the minimum Toffoli gate-count implementation, which has Toffoli count and depth equal to $40N^3$ for an N bit number [43]. We then determine the minimum possible space-time overhead per magic state for this task and also the smallest possible factory, in terms of physical qubits, that can produce all the magic states necessary while keeping up with a time-optimal quantum computation [14].

The smallest possible factory is not necessarily the most space-time efficient factory possible: these factories tend to use larger k blocks which can make the factory formidably vast (see Fig. 2) but able to produce more qubits in the same time as lower-k protocols. However, these may well produce magic states much faster than required by the fastest possible implementation of the algorithm. Figure 6 shows that it is possible to use fewer qubits than required by the time-optimal implementation and perform your computation at a reduced speed. Equally, it is, of course, possible to increase the size of the factory to produce magic states at a faster rate. In this case, though, the extra qubit overhead is effectively wasted unless the computer is performing many calculations in parallel. We find that all the magic states required for a time-optimal factorization of a 1000-bit number can be produced with a surface code magic-state factory of 5.6 million "data qubits" if the infidelity of operations on physical qubits is 10^{-4}. This is based on the cost of d^2 physical qubits to store the information in the rotated lattice surface code [29]. However, for many architectures this number must be doubled to provide ancillas responsible for syndrome extraction. In this case the physical qubit overhead would be ∼11 million.

A summary of the time and space overheads for some example Shor tasks can be found in Table I. We selected Shor's algorithm as a benchmark as the number of non-Clifford gates required has been well studied and these results have been used in previous analyses. We note that due to the large resource overhead that we have demonstrated, early quantum computers may focus on other problems, particularly in

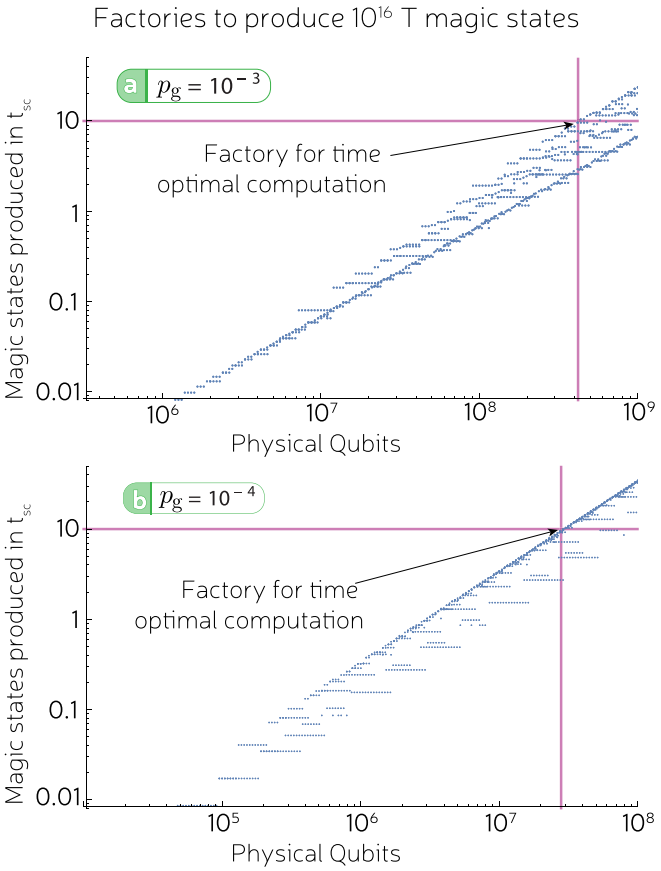

FIG. 6. To produce T magic states at a "time-optimal" rate the factory must, on average, produce states at a rate equal to the fastest rate at which they can be used in sequence by an algorithm. In this paper, where we have assumed that $t_{sc} = 10t_{meas/ff}$, this means ten magic states must be produced on average every t_{sc} to keep up with the time-optimal implementation of the algorithm. As this figure makes clear, it is indeed possible to produce a given number of magic states faster (slower) than this using more (fewer) qubits. Each data point represents one possible magic-state factory given the input error and target number of 10^{16} T magic states; only the factories near the boundary between possible and impossible factories are shown. Not shown in the lower right of each graph are myriad other possible factories of lower rate and higher overhead. As seen clearly in (a), doubling the size of the factory can allow one to more than double the rate of magic-state production when the larger factory allows the use of the higher-k (and therefore higher-rate) block codes. This point is less evident in (b), where only two rounds of distillation can be sufficient and the higher-k block codes are not necessarily optimal.

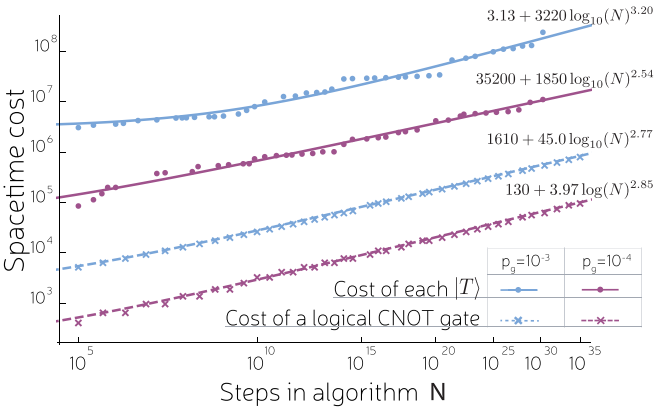

FIG. 7. The scaling of resource cost in qubit rounds per magic state is not worse than that of the surface code. This corroborates the predictions of Raussendorf *et al.* [28] and suggests that the asymptotic overhead scaling $\sim O(\log(N)^3)$ of the surface code is applicable to *universal* fault-tolerant computing with gate counts in the regime of practical interest. Lines are fitted functions of the form $\mathcal{V}(p_g, N) = a\log(N)^b + c$.

quantum chemistry. A recent analysis of some such problems [44] demonstrates that they are solvable with overheads lower than we find here, albeit that work makes more optimistic assumptions of gate times and fidelities.

Of significant interest is how the cost of magic-state distillation compares to the surface code overhead. Raussendorf *et al.* (see Sec. 6.2 of Ref. [28]) were the first to note that using balanced investment in a magic-state distillation leads to a constant factor overhead compared to the CNOT gate. Our numerical results in Fig. 7 show a ratio between T-gate cost and CNOT cost in the range of ∼150–310 when $10^{10} < N < 10^{30}$. This ratio is much smaller than estimated by Raussendorf *et al.,* who did not make use of Bravyi-Haah distillation routines. A similar ratio can be extracted from the data tables provided in Ref. [3], although the numbers are not directly comparable. For instance, we also count the space-time volume due to idle qubits, while they wait for distillation circuits to succeed.

VII. CONCLUSIONS

We proposed the notion of gauge-MSD, which is faster and uses fewer ancillas than previous realizations of Bravyi-Haah magic-state distillation. We further introduced module checking as a means to exploit correlations and found it gave an additional factor of ∼3 reduction in some parameter regimes. Fowler *et al.* [3] considered realizing Bravyi-Haah using braiding and found that Bravyi-Haah offered only a modest factor ∼3 improvement over the first magic-state proposal that used Reed-Muller codes [1]. Therefore, our gains are comparable to, and build upon, other advances in the field. The work of Bravyi and Haah predicted a much greater improvement because they quantified cost by the expected conversion efficiency of raw to high-fidelity magic states. In a fully costed analysis, which we perform here, error correction costs overwhelm and dominate the cost of magic-state factories. We saw that an efficiently designed factory using balanced investment is entirely limited by the surface code cost, so refinement in distillation protocols can offer only constant factor improvements.

It is likely that this analysis represents an overestimate of the space-time overhead of the implementations of the distillation circuits we describe. We have assumed the need for d rounds of surface code measurements after every two-qubit gate. However, it is not clear that this is necessary when performing transversal gates. In the implementation of Bravyi-Haah (see Fig. 1) it may prove feasible to perform d rounds of error correction only after, say, the completion of each of the four steps described, reducing the time overhead from $11t_{sc}d$ to $4t_{sc}d$ [45]. Our cost analysis here could thus be further developed by considering the effect on the performance of the underlying surface code if multiple transversal operations were performed between rounds of error correction. However, such simulations lie beyond the scope of this paper.

Three-dimensional (3D) gauge color codes [7,25] and other recent ideas do not require magic states. But they have their own hidden costs. For 3D gauge color codes, spatial overheads scale as $\sim O(\log(N)^3)$, and time overhead scales as $O(1)$. Using balanced investment and surface codes, we see similar asymptotic scaling of resources. However, current evidence indicates an order of magnitude worse phenomenological threshold for color codes [11,12] compared to the phenomenological threshold for the surface code. Although a full circuit-based threshold has not yet been determined, it is unlikely to challenge that of the surface code due to the higher weight stabilizers required. This points towards 3D color codes requiring physical error rates below 0.1%. Resource costs are heavily influenced by proximity to threshold, so 3D color codes seem to require significantly lower physical error rates before they can start to compete with surface codes augmented by magic states. Therefore, with current technology and fidelities, known schemes for avoiding magic states are a false economy. An additional benefit of the magic-state paradigm is that it can also eliminate the additional burden of gate-synthesis costs by preparing exotic magic states [40,41,46–49].

ACKNOWLEDGMENTS

E.C. is supported by the EPSRC (Grant No. EP/M024261/1). We thank M. Howard for comments on the manuscript. We acknowledge support from the EPSRC National Quantum Technology Hub in Networked Quantum Information Technologies in facilitating this collaboration. The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work.

APPENDIX A: COMPARISON WITH BRAIDING

Here we discuss how our results compare with prior work on braiding defects in the surface code. It has been shown that Bravyi-Haah can be realized in constant time [3] assuming the architecture supports constant time implementations of multitarget CNOT gates (in time $t_{\mathrm{MT-cnot}}$). This has time cost

$$t^{(2)}_{\mathrm{block}} = 12t_{\mathrm{MT-cnot}} + t_{\mathrm{A}} + t_{\mathrm{prep}} + t_{\mathrm{inject}} + t_{\mathrm{measure}}, \qquad \text{(A1)}$$

where t_{inject} is the time to inject a magic state into the circuit and we can infer $t_{\mathrm{inject}} = t_{\mathrm{cnot}} + t_{\mathrm{measure}}$.

In the standard circuit model, multitarget CNOT gates do not take constant time to implement. In the braiding picture, using ancillary defects, this is possible. The "time" cost of braiding 12 such gates is $12 \times 1.25 \times d \times t_{\mathrm{sc}}$. The total space-time cost was reported as $(96k + 216)$ so-called plumbing pieces, which converts into $(\frac{5}{4})^3(96k + 216)$ qubit rounds. It has recently been shown that lattice surgery also supports multitarget CNOT gates in constant time [50]. Although the Bravyi-Haah protocol was not considered in this setting, one can infer a lattice surgery time cost also scaling with $\sim 12d$, but the qubit cost is not currently known.

The above discussion implies a modest space-time saving of using gauge-MSD in a distributed architecture rather than braiding in a nearest-neighbor picture. We remark that gate times and qubit expense will vary on a much greater scale between different hardware platforms. In particular, the long-range gates of distributed schemes are often much slower, with photonic protocols impeded by photon loss and the potential need for entanglement purification [31–38].

APPENDIX B: FORMAL TOOLS

1. Stabilizer Bravyi-Haah codes

Let us begin with some basic concepts from code theory and related notation. Stabilizer codes are subspaces described

by an Abelian group called the stabilizer $\mathcal{S}$, which is a subgroup of the Pauli group. The projector onto the code space is $\Pi \propto \sum_{s\in\mathcal{S}} s$, so that $s\Pi = \Pi$ for all $s \in \mathcal{S}$. There always exists a minimal set of operators $\{S_1,S_2,...S_m\}$ that generates the group, which we denote by $\mathcal{S} = \langle S_1,S_2,\ldots,S_m\rangle$. For the Calderbank-Shor-Steane (CSS) codes, these generators can be chosen so that they are all either X type or Z type, as we define shortly. If g is a binary vector of length n, we use $Z[g] = \otimes^n_{j:[g]_j=1} Z_j$ and, similarly, $X[g] = \otimes^n_{j:[g]_j=1} X_j$. We say $Z[g]$ are Z-type operators and $X[g]$ are X-type operators. Therefore, a CSS code has generators $\mathcal{S} = \langle Z[f_1],\ldots,Z[f_a],X[g_1],\ldots,X[g_b]\rangle$. Commutation of $X[g]$ and $Z[f]$ is equivalent to $(f,g) = 0$, where we use the inner product $(f,g) := fg \pmod 2$.

Bravyi and Haah introduced the notion of a G matrix which is a binary matrix composed of two submatrices, G_1 and G_0. We label the rows of G as $\{g_1,\ldots,g_b,g_{b+1},\ldots g_m\}$, where the rows $\{g_1,\ldots,g_b\}$ belong to G_0 and the rows $\{g_{b+1},\ldots,g_m\}$ belong to G_1. These matrices define a CSS code as follows. The rows of G_0 are some set $\{g_1,\ldots,g_b\}$ that specifies the X-type generators $\{X[g_1],\ldots,X[g_b]\}$. The Z-type generators are given less explicitly. Denote $\mathcal{G}^\perp$ as the binary vector space orthogonal to both G_0 and G_1, that is, $\mathcal{G}^\perp := \{g : (f,g) = 0; \forall\, f \in G_0,G_1\}$. Note that Bravyi and Haah required that $G_0 \subset \mathcal{G}^\perp$. We define $G^\perp$ as some (minimal) matrix with rows $\{f_1,\ldots,f_a\}$ that generate the group $\mathcal{G}^\perp$ under row-wise modular addition. This defines the Z-type generators $\{Z[f_1],\ldots,Z[f_b]\}$. Note that there exist many different choices for $G^\perp$, which all result in the same CSS code with Z-type generators $\{Z[f_1],\ldots,Z[f_a]\}$.

This completes the description of the stabilizer code space, although we also need to know how information is stored within the subspace. We have that the X operator for the kth logical qubit is $X[g_k]$, where g_k is a row of G_1, so $b+1 \leqslant k \leqslant m$. These are representatives of the logical operators, with equivalent logical operators differing by only a stabilizer. For Bravyi-Haah protocols all rows in G_1 have odd weight, so we may also take the Z operator for the kth logical qubit as $Z[g_k]$, where g_k is the kth row of G_1.

2. Subsystem Bravyi-Haah codes

Given a G matrix, we define a subsystem code with stabilizer $\tilde{\mathcal{S}} := \langle Z[g_1],\ldots,Z[g_b],X[g_1],\ldots,X[g_b]\rangle$, where $\{g_1,\ldots,g_b\}$ are the rows of G_0. Notice that now there are equal numbers of X and Z stabilizers, and they both correspond to the rows of G_0. Bravyi and Haah considered a class of matrices obeying triorthogonality conditions, which require that $G_0 \subset \mathcal{G}^\perp$. Therefore, we have that $\tilde{\mathcal{S}} \subset \mathcal{S}$, with the subsystem code having strictly fewer stabilizers than the original Bravyi-Haah code. We denote the subsystem projector as $\tilde{\Pi} \propto \sum_{s\in\tilde{\mathcal{S}}} s$. We further take the logical operator for the subsystem code to be identical to those of the original Bravyi-Haah code. This leaves some degrees of freedom as neither stabilizers nor logical operators. We define the gauge group $\mathcal{S}_g := \langle Z[f_1],\ldots,Z[f_a],X[f_1],\ldots,X[f_a]\rangle$, where $\{f_1,\ldots f_a\}$ are rows of $G^\perp$. Notice that $\mathcal{S}_g$ contains $\mathcal{S}$ by virtue of $G_0 \subset \mathcal{G}^\perp$.

Let us recap. Our subsystem code is defined by its stabilizer $\tilde{\mathcal{S}}$ and gauge group $\mathcal{S}_g$, whereas the original Bravyi-Haah code has stabilizer $\mathcal{S}$, and these groups satisfy $\tilde{\mathcal{S}} \subset \mathcal{S} \subset \mathcal{S}_g$. However, $\mathcal{S}_g$ is inflated in size compared to $\mathcal{S}$ and is no longer Abelian. Furthermore, one can verify that $\mathcal{S}_g$ does not contain any logical operators as follows. First, note that Bravyi and Haah use triorthogonal (also called triply even) matrices where for any $f,g \in G$ we have $(f,g) = 1$ if and only if $f = g$ and $f \in G_1$.

As logical operators we take $X[l]$ and $Z[l]$ for each l in G_1. From the triorthogonality of G we see that $X[l]$ and $Z[l]$ anticommute, but $X[l]$ and $Z[l']$ commute when $l \neq l'$. To properly describe a subsystem code, where measuring the gauge operators does not damage the logical qubits, we require that the logical operators are not elements of the gauge group $\mathcal{S}_g$. Recall that the gauge group is defined by vectors that reside in the dual code $\mathcal{G}^\perp$. Therefore, every gauge operator must have a vanishing inner product with every row in $\mathcal{G}$. However, $l \in \mathcal{G}$, and $(l,l) = 1$, so l is not in the dual space, and the logical operators are not gauge operators. This completes our proof that the logical operators indeed lie outside the gauge group.

APPENDIX C: REALIZING THE BRAVYI-HAAH PROTOCOLS

There are many routes to realizing magic-state distillation. Assuming perfect Clifford operations, different realizations suppress errors equally but differ in terms of temporal depth and required ancillas. Many of these potential realizations have only been sketched, without a complete assessment of resources involved. Here we introduce a method particularly suitable for architectures implementing logical gates via transversal operations or lattice surgery [29]. Conceptually, we are inspired by notions of subsystem codes and gauge-fixing techniques and so call our approach gauge-MSD. We consider only even k with $k \in \{2,6,\ldots,4m+2,\ldots\}$ as then the Bravyi-Haah codes have transversal T gates. For $k \in \{0,4,8,\ldots,4m,\ldots\}$, the Bravyi-Haah codes have transversal T gates only up to a nonlocal Clifford correction.

1. Outline of protocol

Here we present an outline of gauge-MSD, with details of how to realize multiqubit Pauli measurements postponed until the next section. First, we specify some notation. Refer back to Appendix B for definitions of the G matrix and $G^\perp$ matrix. Let R be a binary matrix such that $G^\perp \cdot R = \mathbb{1} \pmod 2$, which is ensured to exist by virtue of the fact that $G^\perp$ is full rank. Furthermore, let M be a binary matrix such that $M \cdot G^\perp = G_0 \pmod 2$, which must exist since G_0 is in the span of $G^\perp$. Explicit examples of $G^\perp$, R, and M will be given in the next section. Measurement outcomes will be recorded in binary: 0 for $+1$ eigenvalues and 1 for -1 eigenvalues. Let $\mathcal{O}$, $\mathcal{X}$, and $\mathcal{Z}$ be disjoint sets:

$$\begin{aligned}\mathcal{O} &= \{6+3j \,|\, j = 1,2,\ldots k\},\\ \mathcal{X} &= \{1,2,3\},\\ \mathcal{Z} &= \{4,5,6,7,8,7+3j,8+3j \,|\, j = 1,2,\ldots,k\}. \end{aligned} \quad \text{(C1)}$$

Associated with these sets are binary matrices that allow us to compute Pauli corrections; H_Z is a k-by-$|\mathcal{X}|$ matrix and H_X is

a k-by-$|\mathcal{Z}|$ matrix as follows:

$$H_Z = \begin{array}{c} \begin{array}{cccccccccccc} 4 & 5 & 6 & 7 & 8 & 10 & 11 & 13 & 14 & 16 & 17 & \dots \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \dots \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & \\ & & & & & \vdots & & & \vdots & & & \end{pmatrix} \end{array} \tag{C2}$$

$$H_X = \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ & \vdots & \end{pmatrix} \end{array} \tag{C3}$$

where the numbers above the columns correspond to the elements in sets $\mathcal{Z}$ and $\mathcal{X}$. Last, we will also make use of a k-by-dim($G^\perp$) matrix denoted Q that satisfies $G_1 = W + QG^\perp$, where W has $W_{j,6j+3} = 1$, $W_{j,1} = W_{j,2} = 1$ and is zero everywhere else. An example Q is given in the next section.

We now state the protocol:

(1) Measure all $Z[f_k]$, where f_k is the kth row of $G^\perp$, recording outcomes as $\mu = (\mu_1, \mu_2, \dots, \mu_k)$.

(2) Apply $A[w]$, where $w = R\mu \pmod 2$.

(3) Measure all $X[f_k]$, where f_k is the kth row of $G^\perp$, recording the outcome as $\gamma = (\gamma_1, \gamma_2, \dots, \gamma_k)$.

(4) Declare success if $M\gamma = (0, \dots, 0) \pmod 2$ and failure otherwise.

(5) Measure qubits in set $\mathcal{X}$ in the X basis, recording outcomes as m_X. Simultaneously, measure qubits in set $\mathcal{Z}$ in the Z basis, recording outcomes as m_Z.

(6) Qubits in $\mathcal{X}$ and $\mathcal{Z}$ are discarded, while qubits in set $\mathcal{O}$ are retained as output as qubits $(1, 2, \dots, k)$.

(7) Apply Pauli corrections $X[H_Z m_Z]Z[H_X m_X + Q\gamma]$, or update the Pauli frame accordingly.

After steps 1 and 2, the state is deterministically projected by $\Pi_Z \propto \sum_{s \in \mathcal{S}_Z} s$, where $\mathcal{S}_Z := \langle Z[f_1], \dots, Z[f_a] \rangle$, exactly as in the original Bravyi-Haah protocol. After step 3, the system is an eigenstate of $(-1)^{\gamma_k} X[f_k]$, and we can associate some projector with $\Pi_{X,\gamma}$ with this process. The postselection in step 4 ensures the system is an eigenstate of $+X[g_k]$, where g_k is the kth row of $M \cdot G^\perp$. Since we defined M such that $M \cdot G^\perp = G_0$, we have that g_k are the rows of G_0. It follows that $\Pi_{X,\gamma} = \Pi_X \Pi_{X,\gamma}$, where Π_X is the projector for the X stabilizer of the Bravyi-Haah stabilizer code. Combining steps 1–4, we have the projections

$$\Pi_{X,\gamma} \Pi_Z = \Pi_{X,\gamma} \Pi_X \Pi_Z = \Pi_{X,\gamma} \Pi, \tag{C4}$$

where $\Pi = \Pi_X \Pi_Z$ is the full projector onto the Bravyi-Haah stabilizer code space.

We have obtained the desired Π projection required by the Bravyi-Haah protocol. But we have picked up an additional $\Pi_{X,\gamma}$. This additional projection results from measuring some gauge operators of the subsystem variant of Bravyi-Haah. Thus, the logically encoded qubits are unharmed, but there has been a change in the gauge degrees of freedom.

In step 5, we perform single-qubit measurements to isolate the k logical qubits from the n qubits in the subsystem code. Recall that in the original presentation of Bravyi-Haah we have logical operators $Z[l_j]$ and $X[l_j]$ for the jth logical qubit, where l_j is the jth row of G_1. The measurements localize these logical operators onto single qubits, up to a Pauli correction depending on the measurement outcomes. That is, the measurements cause the state to become stabilized by some new $\pm Z[w]$ operators, and every logical $Z[l_j]$ can be multiplied by these operators to obtain a single-qubit $\pm Z$ operator acting on the $(6 + 3j)$th qubit. It is straightforward to verify that the $\pm$ sign is corrected to $+$ by the Pauli correction $X[H_Z m_Z]$. To show that X logical operators are also localized on a single output qubit, we first multiply $X[l_j]$ by X-type gauge operators until it acts on qubits 2, 3, and $(6 + 3j)$. Measuring qubits 1, 2, and 3 completes the localization of $X[l_j]$ onto the $(6 + 3j)$th qubit. The required Pauli operator is now $X[H_Z m_Z + Q\gamma]$, where there is also some dependence on the eigenvalues of gauge operators obtained in step 3.

2. Implementing Pauli measurements in minimum depth

Implementing the protocol requires a set of Z measurements, followed by a set of X measurements. In the original standard implementation the X measurements involved many qubits and so were difficult to implement. However, the previous section shows that the difficult X measurements can be replaced with lower-weight measurements mirroring the Z measurements performed. The complexity of these measurements depends on the row weights of $G^\perp$. The matrix $G^\perp$ must generate the space $\mathcal{G}^\perp$, but there is some freedom in how we choose the generating rows. In the work of Bravyi-Haah, $G^\perp$ was only implicitly defined (as the dual of G), leaving unclear how much time it would take to implement the required measurements. It is desirable that $G^\perp$ is as sparse as possible to minimize the resource overheads. Therefore, we wish to find a very sparse $G^\perp$. We found a family of $G^\perp$ matrices where all rows, except one, are weight 4 and the single exception has weight $k + 2$. We present the exact form of this $G^\perp$ for $k = 2$, along with the associated R, M, and Q matrices used in the previous section:

$$G^\perp = \left(\begin{array}{cccccccc|ccc|ccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{array}\right), \tag{C5}$$

$$R = \begin{pmatrix} 1&0&0&0&1&1&0&0&0\\ 0&1&0&1&0&1&0&0&0\\ 0&0&1&1&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ 0&0&1&1&1&1&0&0&1\\ 0&0&1&1&1&1&1&0&1\\ 0&0&1&1&1&1&1&1&1 \end{pmatrix}, \quad Q = \begin{pmatrix} 1&1&0&0&0&1&0&0&0\\ 1&1&0&0&0&1&0&1&0 \end{pmatrix},$$

$$M = \begin{pmatrix} 0&1&0&1&1&1&1&1&0\\ 0&0&1&1&1&1&0&1&0\\ 1&1&1&1&1&1&0&0&0 \end{pmatrix}.$$

A *Mathematica* script for generating $G^\perp$ for any k is provided in the Supplemental Material [51], which also verifies that $G^\perp$ is full rank and dual to G. As one further example, we find for $k = 6$ that

$$G^\perp = \left(\begin{array}{cccccccc|ccc|ccc|ccc|ccc|ccc} 1&0&0&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&1&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&1&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1\\ 0&0&1&0&0&0&1&0&1&0&0&1&0&0&1&0&0&1&0&0&1&0&0&1&0&0&1 \end{array}\right). \tag{C6}$$

Notice that the bottom row has weight exceeding 4. The measurements corresponding to weight-4 rows can be implemented with each measurement using a single ancilla in the $|+\rangle$ state and four entangling gates (control Z or control X depending on the measurements). Therefore, for these measurements it is possible that all these gates can be realized in four time steps while respecting that a qubit can be involved in only a single entangling gate at a time. Unfortunately, there is a single row of $G^\perp$ with weight $k+2$, so using a single ancilla to perform this measurement would result in a growing time cost with k. Therefore, for this single measurement we make use of a cat state $|0\rangle^{\otimes k+2} + |1\rangle^{\otimes k+2}$ so that the entangling gates can be performed in parallel. The cat state itself is constructed by merge operators on $|+\rangle^{\otimes k+2}$ qubits. The merge operations project onto $|00\rangle\langle 00| + |11\rangle\langle 11|$ or $|01\rangle\langle 01| + |10\rangle\langle 10|$ subspaces and so commute with the control gates, so the cat state can be built concurrently with the control gates. This opens the possibility of realizing each round of measurements in four times steps but depends on whether the entangling gates can be scheduled in an economical manner. The scheduling problem is equivalent to a graph coloring problem, and we find that it can be solved in four time steps (e.g., using four colors) as in Fig. 1. We independently confirmed this using an automated solver of the edge colorability problem for k up to 40; see supplementary *Mathematica* script for details.

APPENDIX D: REED-MULLER CONNECTIVITY

The usual form of the 15-qubit punctured Reed-Muller code G matrix is

$$G_{\mathrm{RM}} = \left(\begin{array}{ccccccccccccccc} 1&1&1&1&1&1&1&1&0&0&0&0&0&0&0\\ 1&1&1&1&0&0&0&0&1&1&1&1&0&0&0\\ 1&1&0&0&1&1&0&0&1&1&0&0&1&1&0\\ 1&0&1&0&1&0&1&0&1&0&1&0&1&0&1\\ \hline 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \end{array}\right). \tag{D1}$$

For our purposes an efficient choice of dual is

$$G_{\mathrm{RM}}^{\perp} = \begin{pmatrix} 0&0&0&0&1&1&1&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&1&1&1&0&0&0\\ 0&0&0&0&0&0&0&0&1&1&0&0&1&1&0\\ 0&0&0&0&0&0&0&0&1&0&1&0&1&0&1\\ 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0\\ 1&1&0&0&1&1&0&0&0&0&0&0&0&0&0\\ 1&0&1&0&1&0&1&0&0&0&0&0&0&0&0\\ 0&1&1&0&0&0&0&0&0&1&1&0&0&0&0\\ 0&0&1&0&1&0&0&0&0&0&1&0&1&0&0\\ 1&0&0&0&1&0&0&0&1&0&0&0&1&0&0 \end{pmatrix}, \tag{D2}$$

which has a maximum row weight of 4 and a max column weight of 5. There is a five-colorable graph associated with this matrix shown in Fig. 8, similar to what we found for the Bravyi-Haah protocol. It is well known that the Reed-Muller code can also be viewed as a subsystem code, so the Pauli-X measurements can clone the pattern of the Pauli-Z measurements. This approach yields the realization shown in Fig. 8.

APPENDIX E: TOFFOLI PROTOCOL

The Toffoli protocol converts eight T states into one Toffoli state,

$$|\mathrm{Tof}\rangle = \frac{1}{\sqrt{8}} \sum_{a,b,c\in\{0,1\}} (-1)^{abc} |a,b,c\rangle. \tag{E1}$$

It has previously been described in the circuit picture with its performance found by brute-force counting of errors [26,27]. Here we consider an equivalent protocol (with the same performance when using block checking) but with the G matrix formalism of the Bravyi-Haah protocols. The G matrix achieving this is simply

$$G_{\mathrm{Tof}} = \left(\begin{array}{cccccccc} 1&1&1&1&0&0&0&0\\ 1&1&0&0&1&1&0&0\\ 1&0&1&0&1&0&1&0\\ \hline 1&1&1&1&1&1&1&1 \end{array}\right), \tag{E2}$$

where proof that this distills was given in Refs. [40,41]. Here we show in Fig. 9 how to convert this abstract protocol to a realization with low space-time overhead.

APPENDIX F: PROOF OF ERROR TRACKING

When iterating distillation protocols, we have a collection of inputs which may have been outputs of an earlier round. We illustrate the structure in Fig. 2, where we introduce the terminology of branches and modules. Our proof proceeds by considering how an individual module maps probability distributions over its inputs to a probability distribution over outputs. If a level-l module uses many $n_l \to k_l$ blocks, then it requires n_l branches of inputs from earlier rounds. If each branch carries N_l qubits, then we need N_l instances of the $n_l \to k_l$ protocol so that the whole module maps $N_l n_l \to N_l k_l$ qubits. The size of N_l equals the product of the k_l values for all earlier rounds, which can be verified by simply counting back through the tree.

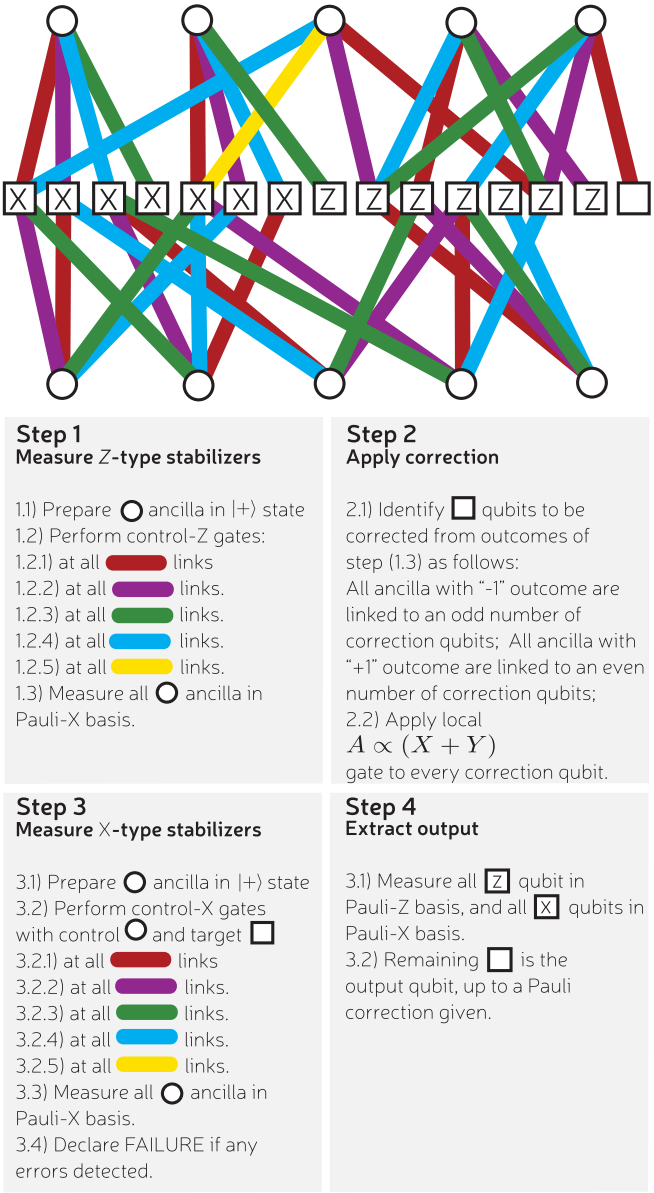

FIG. 8. Squares represent magic states to be distilled, and circles are ancillas used to implement measurements in Reed-Muller. Edges show required hardware connections and associated required control-phase gates. Edges show time ordering of control-phase gates, e.g., red, purple, green, blue, and gold (dark to light gray), demonstrating the required entanglement can be established in five time steps. Sometimes we call this the graph representing G_z^{RM}.

We use $x^{(j)} \in \mathbb{Z}_2^N$ to denote the error distribution of the input contribution from the jth branch and use probability $\Pr(x^{(j)})$ for the probability of this event. For now we take this probability as given and note only that for a block that quadratically suppresses noise, we have that $\Pr(x^{(j)} \neq 0)$ is, to first order, proportional to ϵ^{2^l} after l rounds of distillation. If a single block of an $n_l \to k_l$ protocol is described by a matrix G, then N copies within a module are described by $\mathcal{G} = G \otimes \mathbb{1}_N$, where $\otimes$ is the tensor product and $\mathbb{1}_N$ is the identity matrix of dimension N. On the first round, $N = 1$, so we simply have $\mathcal{G} = G$. Before proceeding we must clarify

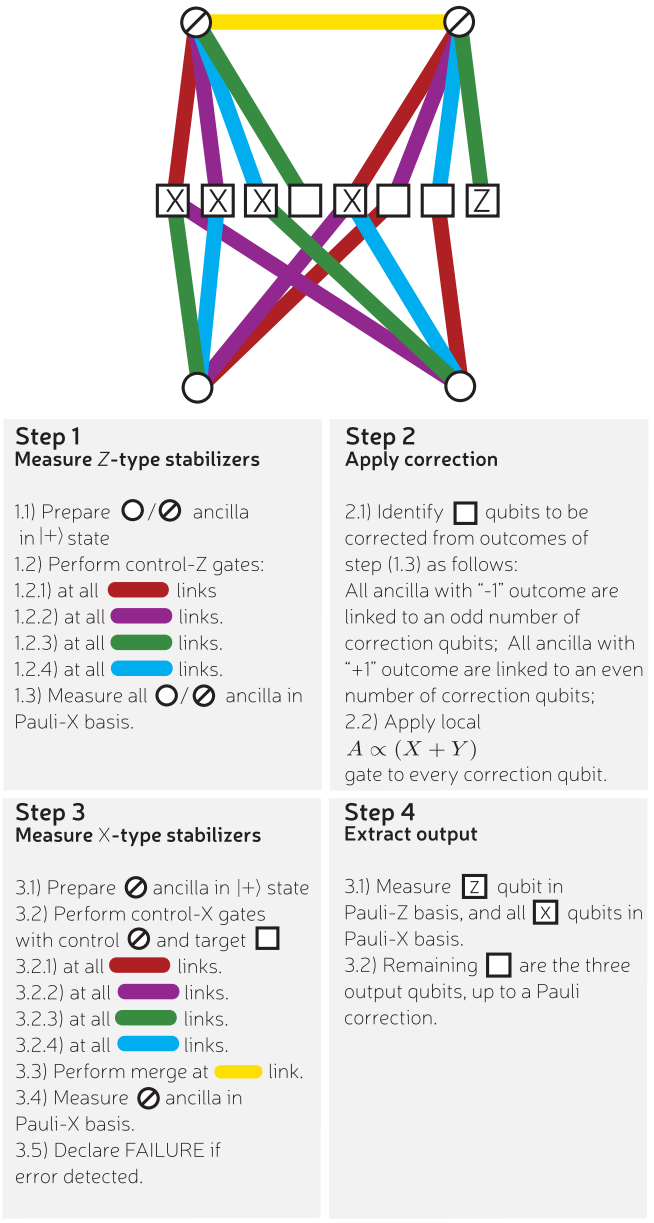

FIG. 9. Squares represent magic states to be distilled, and circles are ancillas used to implement the stabilizer measurements which project the T states into the Toffoli state. As before, edges show required hardware connections and associated control-phase gates. Edges show time ordering of control phase gates, e.g., red, purple, green, blue, and gold (dark to light gray), demonstrating the required entanglement can be established in four time steps for the Z stabilizers and five time steps for the single X stabilizer, which utilizes just two ancillas. We call this the graph representing G_z^{Toff}.

how this tensor product structure relates to the input strings x. We define $\delta^{(j)}$ as a length-n_l binary vector with entries 1 in the jth location and 0 everywhere else. It follows that $x = \sum_j (\delta^{(j)} \otimes x^{(j)})$, so that $\mathcal{G}x = \sum_j (G\delta^{(j)} \otimes x^{(j)})$. This ordering ensures that no two qubits from the same branch collide into the same block, which must be prevented because of correlations within branches. We call this canonical ordering, and it is assumed throughout. Other orderings exist that prevent such collisions, such as an arbitrary permutation of qubits within any branch. Potentially, such permutations could perform better or worse than the canonical choice, leaving room for further optimization. We continue with the canonical choice as it is particularly natural and amenable to analysis.

After a module passes module checking, the output branch has errors distributed as

$$\Pr_l(y) = \frac{1}{\mathrm{P}_{\mathrm{suc},l}} \sum_{\{x:\mathcal{G}_0 x=0,\mathcal{G}_1 x=y\}} \Pr(x), \tag{F1}$$

where the denominator is a normalization constant accounting for the success probability

$$\mathrm{P}_{\mathrm{suc},l} = \sum_y \sum_{\{x:\mathcal{G}_0 x=0,\mathcal{G}_1 x=y\}} \Pr(x). \tag{F2}$$

For our proof, it is useful to work with unnormalized probabilities that we write as

$$P_l(y) = \sum_{\{x:\mathcal{G}_0 x=0,\mathcal{G}_1 x=y\}} P_{l-1}(x), \tag{F3}$$

where we will renormalize later. Herein, we focus on the smallest weight errors that can lead to a particular y. The no-error case ($y = 0$) occurs when the initial states had no errors, so

$$P_l(0) \simeq (1-\epsilon)^{m_l}, \tag{F4}$$

where m_l counts the total number of raw qubits needed for l rounds of distillation, namely, $m_l = \prod_{j=1,\ldots,l} n_j$. For protocols quadratically suppressing noise, 2^l is the smallest number of errors that can evade detection. Therefore, we take the approximation

$$P_l(y) \simeq C_l(y)\epsilon^{2^l}(1-\epsilon)^{m_l-2^l}, \tag{F5}$$

where $C_l(y)$ counts the number of different weight-2^l errors that lead to output y. For one round of distillation this is simply

$$C_1(y) = \eta(y). \tag{F6}$$

Recall that $\eta(y)$ was defined back in Definition 1 as a function that counts precisely the number of weight-2 errors that lead to a particular output.

Finding $C_l(y)$ for higher levels ($l > 1$) is more involved. The relation between input and output error strings is $y = \mathcal{G}_1 x$ (assuming $\mathcal{G}_0 x = 0$), so x vectors of weight 2 can result in y vectors of a variety of weights, potentially increasing the weight, so some care is needed.

Consider a module (on some $l > 1$ level) where some of the incoming branches contain errors. The probability of two branches a and b containing errors is

$$\begin{aligned} &P_{l-1}(x^{(a)})P_{l-1}(x^{(b)})P_{l-1}(0)^{n_l-2} \\ &\quad\simeq C_{l-1}(x^{(a)})C_{l-1}(x^{(b)})\epsilon^{2^l}(1-\epsilon)^{m_l-2^l}. \end{aligned} \tag{F7}$$

These errors may contribute to undetected errors leaving the module. Fewer or more branch failures do not provide leading-order contributions. Consider when only one branch contains any errors. After this branch is split up and fed into different blocks, each $n_l \to k_l$ block can contain at most one error, so it will be detected. If $t > 2$ branches contain an error, the

probability will be weighted by $\epsilon^{t*2^{(l-1)}}$, which for small ϵ is a rare process compared to two failed branches.

Furthermore, with two erroneous branches a and b, we can further deduce that either $x^{(a)} = x^{(b)}$ or the error will be detected. To see this, we begin by noting that errors will be undetected only if $(G_0 \otimes \mathbb{1}_N)x = 0 \pmod 2$, and we have

$$(G_0 \otimes \mathbb{1}_N)x = (G_0\delta^{(a)}) \otimes x^{(a)} + (G_0\delta^{(b)}) \otimes x^{(b)}. \tag{F8}$$

Both $G_0\delta^{(a)}$ and $G_0\delta^{(b)}$ are columns of G_0 and so are nonzero. Since no terms are zero, it can vanish mod 2 only if both $G_0\delta^{(a)} = G_0\delta^{(b)}$ and $x^{(a)} = x^{(b)}$. This is a central point of the proof, so we will give a second explanation of what is happening here. Note that if $x^{(a)} \neq x^{(b)}$, then without loss of generality $x^{(a)}$ had an error in at least one location that is absent from $x^{(b)}$. If the error's location is t, then the tth distillation block will receive an input with only one error and must detect it. We conclude that the dominate source of errors is from the identical failure of pairs of branches.

To simplify notation, let $u = \delta^{(a)} + \delta^{(b)}$ and $f = x^{(a)} = x^{(b)}$, so we have the more concise expression $x = u \otimes f$ for relevant errors, with u being weight 2. Above we noted that an undetected error must satisfy $G_0\delta^{(a)} = G_0\delta^{(b)}$, which is equivalent to $G_0u = 0 \pmod 2$. The output from such an error is $y = (G_1 \otimes \mathbb{1}_N)(u \otimes f) = (G_1u) \otimes f$. Therefore, we can now deduce that

$$P_l(v \otimes f) = \sum_{\{u:G_0u=0,G_1u=v\}} P_{l-1}(f)^2 P_{l-1}(0)^{n_l-2}$$

by counting over all u that lead to the same v. Therefore,

$$C_l(v \otimes f) = \sum_{\{u:G_0u=0,G_1u=v,|u|=2\}} C_{l-1}(f)^2.$$

The summation is over weight-2 vectors u, but the terms are independent of u, so we find that

$$C_l(y) = C_l(v \otimes f) = \eta_l(v)C_{l-1}(f)^2,$$

where we have again used the η notation introduced in Definition 1. For two rounds of distillation, $l = 2$, so $C_{l-1} = C_1$, and we can end the recursion by using Eq. (F6), so that

$$C_2(y) = C_2(v \otimes f) = \eta_2(v)\eta_1(f)^2. \tag{F9}$$

However, if we have more than two rounds, notice that the relevant f will again have the form $f = v' \otimes f'$, so that

$$\begin{aligned} C_l(y) &= \eta_l(v)[C_{l-1}(v' \otimes f')]^2 \\ &= \eta_l(v)[\eta_{l-1}(v')C_{l-2}(f')^2]^2 \\ &= \eta_l(v)[\eta_{l-1}(v')]^2[C_{l-2}(f')]^4 \end{aligned} \tag{F10}$$

and so on until we reach C_1 and can use Eq. (F6) to end the recursion.

From $C_l(y)$ we have a good leading-order approximation of the probability of an error $P_l(y)$. The total (unnormalized) probability of an error is

$$\begin{aligned} B_l &= \sum_{y\neq 0} P_l(y) = \epsilon^{2^l}(1-\epsilon)^{m_l-2^l} \sum_{y\neq 0} C_l(y) \\ &= \epsilon^{2^l}(1-\epsilon)^{m_l-2^l} C_l, \end{aligned} \tag{F11}$$

where we have used the shorthand

$$C_l = \sum_{y\neq 0} C_l(y) = \prod_{j=1}^{l} \left(\sum_v \eta_j(v)^{2^{l-j}} \right). \tag{F12}$$

Equating also $A_l = P_l(0) = (1-\epsilon)^{m_l}$, we find we have reached the expression of Eq. (8). To renormalize the error probability B_l, we simply divide through by the total probability $A_l + B_l$, yielding one equation of Theorem 1. The denominator $A_l + B_l$ represents the success probability of not just a single module succeeding but all events feeding into that module also succeeding. We are actually interested in the success probability conditioned on previous modules being successful, which is $(A_l + B_l)$ divided by $(A_{l-1} + B_{l-1})^{n_l}$. This divider comes from the unconditional success probability of an $(l-1)$-level module, $(A_{l-1} + B_{l-1})$, and the fact that n_l of these feed into an l-level module. This completes the proof of Theorem 1.

APPENDIX G: NUMERICAL SIMULATIONS

1. Brute-force method

In simulating a magic-state factory it quickly becomes apparent that a brute-force method of simulation is inadequate. The "brute-force" method simply involves randomly generating a Boolean string of length $\prod_l n_l$ to describe the input to the magic-state factory and performing calculations of the stabilizers and logical output of each module of the factory. The explicit procedure for a three-round factory is given in Algorithm 1. For each module in round 1, a random input is generated and tested until the stabilizer checks for the module are passed. The logical output of each of these successful modules is saved until enough have been generated to feed into round 2. These outputs are shuffled according to Eq. (F8) before entering round 2. Here the module checking is performed again. If all the module checks are passed, then we again proceed by feeding the logical output of round 2 to round 3, after shuffling. Having reached round 3, we then determine the characteristics of this module by recording where the module check fails and, if it succeeds, whether the output of the module contains a logical (undetected error).

Clearly, a large number of iterations of this protocol are required to obtain reliable statistics for the performance of the factory. For example, our analytic treatment estimates that with a raw magic-state error rate $\epsilon = 0.001$ the rate of undetected logical error of a round-3 module is $\sim 10^{-21}$. As such, successfully simulating this by brute force would require $\gg 10^{21}$ attempts at the algorithm to be made, not just to build statistics but to ensure that enough instances of the third-round module are generated in the first place. We find that the simulation of two-round factories by brute force is achievable for $2 < k < 50$, but to simulate three rounds a different method is required.

2. Rare-event method

An undetected error in the factory's output after three rounds of distillation is a rare event. To simulate these events and gain adequate statistics we must use a method in which we, as far as possible, eliminate the simulations of input

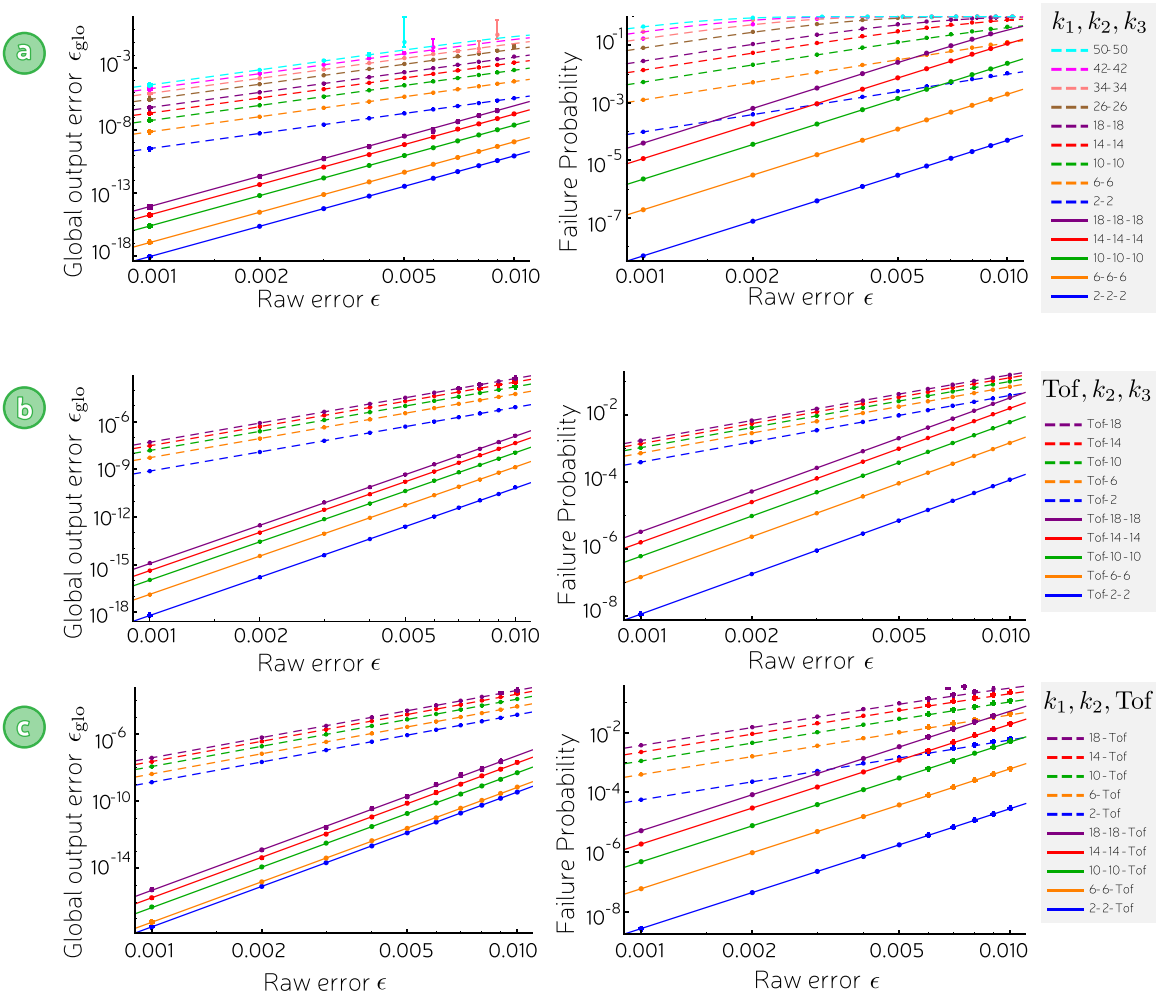

FIG. 10. Numeric vs analytic. (a) Bravyi-Haah block protocols with two and three rounds of distillation. (b) Toffoli protocol followed by one or two rounds of Bravyi-Haah. (c) One or two rounds of Bravyi-Haah followed by the Toffoli protocol. The close correspondence of our analytic treatment of the factory and numerical simulations is demonstrated for two and three rounds of the magic-state distillation with module checking. Points represent simulation data, while the lines are the corresponding analytic estimates as determined by Theorem 1. The protocols are labeled by the k value of each round, which is set to be equal for each round of distillation. The global error, the probability of a logical error anywhere in the output of the factory, is shown. For a "2-2-2" factory this is the probability of an error anywhere in the 8 output magic states; for a "26-26-26" this corresponds to an error anywhere in the 17,576 magic states that this factory produces.

error configurations that we know cannot lead to a logical error. Our chosen method of doing this proceeds as follows. We know that a module has a chance of failing only if at least two of the branches entering that module contain an error. For a module in the third round, we focus on cases where at least two of its input branches contain errors. We do this using a process that can be called preselection, in contrast to postselection. In postselection, we sample from a distribution and reject instances that do not meet a certain criterion. This is not feasible when the criterion (here having at least two corrupt branches) is rare, so we instead construct a new probability distribution conditioned on the criterion being meet. The first step of the algorithm therefore is to first decide how many error-containing branches will enter this module, given that this number is $\geqslant 2$ based on the statistics already gathered for the rate of errors in the output of round-2 modules. Later, we analytically adjust for this process of preselection.

For each of the "corrupt" branches entering round 3, we know that it must originate from a round-2 module which itself had two or more corrupt branches entering it. These branches again would have originated in a round-1 module (equivalently, a block) which had at least two errors fed to it. We can thus greatly reduce the size of the simulation and the number of iterations required by simulating only on a subsection of the factory where these errors have occurred (see Fig. 11). The probability of an undetected error after the first round was determined analytically for Bravyi-Haah by explicitly calculating the weight enumerator

$$W(k,\epsilon) = 2\sum_{m\text{ is odd}}^{k}(1-2\epsilon)^{3m+4} + \sum_{m=0}^{k/2}(1-2\epsilon)^{3(2m)} + 6\sum_{m=0}^{k}(1-2\epsilon)^{2k-m+4} + \sum_{m=0}^{k/2}(1-2\epsilon)^{3(2m)+8}, \tag{G1}$$

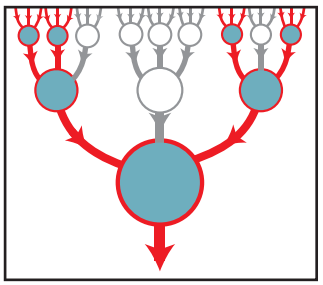

FIG. 11. Rare-event method. When simulating the fictional factory of Fig. 2 with our rare-event method, only the red (darker gray) highlighted parts of the factory are actively simulated, i.e., require computation of the stabilizer outcomes and the logical state of the outputs.

which allows analytic calculation of the global fidelity of the output of a single round of distillation. The corresponding result for the Toffoli protocol is given in Ref. [26].

Having chosen the number of failed branches that need to enter round 3, we attempt to generate these failed branches. We thus simulate the reduced factory, as described above, discarding and repeating each module not only if the stabilizer module check has failed but also when the check is passed and the output does not contain a logical error. Using this method, we eliminate the simulation of a vast number of the possible input error strings, while holding smaller Boolean vectors in memory at all but the last step, the full simulation of the module in round 3. Some pseudocode is presented in Algorithm 2.

This method of simulation allows us to calculate the correlated error rate as follows. A key number is the probability p_{num} that two or more branches leaving round 2 of the factory contain an error:

$$p_{\text{num}} = 1 - \left(1 - \epsilon_{\text{glo}}^{(2)}\right)^{n_3} - n_3 \epsilon_{\text{glo}}^{(2)} \left(1 - \epsilon_{\text{glo}}^{(2)}\right)^{n_3 - 1}, \tag{G2}$$

where $\epsilon_{\text{glo}}^{(2)}$ is the probability that a branch exiting round 2 contains an undetected error, as determined by numerical simulations of a two-round factory.

Using this allows us to determine the success probability and the global error in the output of the round-3 module:

$$p_{\text{suc}}^{(3)} = \left(1 - \epsilon_{\text{glo}}^{(2)}\right)^{n_3} + p_{\text{num}}(a_3 + b_3) \tag{G3}$$

and

$$\epsilon_{\text{glo}}^{(3)} = \frac{b_3 \; p_{\text{num}}}{p_{\text{suc}}} \tag{G4}$$

where $b_3 = \frac{\#\{\text{ERROR}\}}{\#\{\text{SUCCESS}\}+\#\{\text{FAIL}\}+\#\{\text{ERROR}\}}$ is the estimate of the probability of a logical error in the output branch of round 3 from the output of the simulation and $a_3 = \frac{\#\{\text{SUCCESS}\}}{\#\{\text{SUCCESS}\}+\#\{\text{FAIL}\}+\#\{\text{ERROR}\}}$.

We limit our simulations to a limited set of k values, with $k_i = k$ for each simulation. This is an arbitrary choice and simplifies the comparison made in Fig. 10. For three rounds the simulations are limited to low k values, as these can be simulated in a reasonable time frame and adequate statistics can be gathered to infer the output error rate.

Algorithm 1. Brute-force simulation algorithm

select protocol: $\{k_1, k_2, k_{\text{Toff}}\}$
generate number of modules in each round: $\{M_1, M_2, M_{\text{Toff}}\}$ {Round One}
for $i < M_1$ **do**
  randomly generate binary string v length n_1
  measure stabilizers $G_0(k_1).v$
  if stabilizers failed **then** return to line 3
  end if
  calculate logical output of module $v = G_1(k_1)v$
  Append v to list of logical outputs V
end for
shuffle output of round 1 to firewall correlations $V \to V'$ {Round Two}
for $j < M_2$ **do**
  for each block ii in a round-2 module (there are k_1) **do**
    take $(ii \times j)$th string of length n_2 from V': v'
    measure stabilizers $G_0(k_2).v'$
    if stabilizers failed, **than** return to line 1
    end if
    calculate logical output of module $w = G_1(k_2)v'$
    Append w to list of logical outputs W
  end for
end for
shuffle output of round 2 to firewall correlations $W \to W'$ {Round Three}
for $l < k_1 k_2$ **do**
  measure stabilizers $G_0(k_3)w'$
  if stabilizers failed **then return** FAIL
  end if stabilizers passed
  calculate logical output of module $G_1(k_3)w'$
  Search for logical error in output
  if logical error found **then return** ERROR
  else logical error not found **return** SUCCESS
  end if
end for

Algorithm 2. Rare-event simulation algorithm

select protocol: $\{k_1, k_2, k_{\text{Toff}}\}$
generate number of modules in each round: $\{M_1, M_2, M_{\text{Toff}}\}$
generate number of corrupt modules in round 2: N_2 {Round One}
for $i < N_2$ **do**
  generate number of corrupt round 1 modules N_1
  for $j < N_1$ **do**
    randomly generate binary string v length n_1
    measure stabilizers $G_0(k_1)v$
    if stabilizers failed **then** return to line 6
    end if
    calculate logical output of module $G_1 v$
    if there is a logical error, **then** Append v to list of logical outputs V
    else return to line 6
    end if
  end for
  Pad V with zeros so it is length $k_1 n_2$
  shuffle output of corrupt module to firewall correlations $\ldots V \to V'$ {Round Two}
  for each block ii in a round-2 module (there are k_1) **do**

take (ii)th string of length n_2 from V': v'
measure stabilizers $G_0(k_2)v'$
if stabilizers failed, **then** return to line 6
end if
calculate logical output of module $w = G_1(k_2)v'$
if there is a logical error, **then** Append w to list W
else return to line 6
end if
end for
end for {Round Three}
Pad W with zeros so it is length $k_1k_2n_3$
shuffle output of round 2 to firewall correlations $W \to W'$
for $l < k_1k_2$ **do**
measure stabilizers $G_0(k_3)w'$
if stabilizers failed,**then return** FAIL
end if stabilizers passed
calculate logical output of module $G_1(k_3)w'$
Search for logical error in output
if logical error found, **then return** ERROR
else logical error not found **return** SUCCESS
end if
end for

[1] S. Bravyi and A. Kitaev, Phys. Rev. A **71**, 022316 (2005).
[2] C. Jones, Phys. Rev. A **87**, 042305 (2013).
[3] A. G. Fowler, S. J. Devitt, and C. Jones, Sci. Rep. **3**, 1939 (2013).
[4] P. W. Shor, in *Proceedings of the 37th Annual Symposium on Foundations of Computer Science* (IEEE Computer Society Press, 1996), pp. 56–65.
[5] E. Knill, R. Laflamme, and W. H. Zurek, Science **279**, 342 (1998).
[6] R. Raussendorf, J. Harrington, and K. Goyal, Ann. Phys. (NY) **321**, 2242 (2006).
[7] H. Bombin, New J. Phys. **17**, 083002 (2015).
[8] A. Paetznick and B. W. Reichardt, Phys. Rev. Lett. **111**, 090505 (2013).
[9] J. T. Anderson, G. Duclos-Cianci, and D. Poulin, Phys. Rev. Lett. **113**, 080501 (2014).
[10] T. Jochym-O'Connor and R. Laflamme, Phys. Rev. Lett. **112**, 010505 (2014).
[11] B. J. Brown, N. H. Nickerson, and D. E. Browne, Nat. Commun. **7**, 12302 (2016).
[12] S. Bravyi and A. Cross, arXiv:1509.03239.
[13] P. Selinger, Phys. Rev. A **87**, 042302 (2013).
[14] A. G. Fowler, arXiv:1210.4626.
[15] A. M. Meier, B. Eastin, and E. Knill, Quantum Inf. Comput. **13**, 195 (2013).
[16] S. Bravyi and J. Haah, Phys. Rev. A **86**, 052329 (2012).
[17] N. C. Jones, R. Van Meter, A. G. Fowler, P. L. McMahon, J. Kim, T. D. Ladd, and Y. Yamamoto, Phys. Rev. X **2**, 031007 (2012).
[18] H. Anwar, E. T. Campbell, and D. E. Browne, New J. Phys. **14**, 063006 (2012).
[19] E. T. Campbell, H. Anwar, and D. E. Browne, Phys. Rev. X **2**, 041021 (2012).
[20] E. T. Campbell, Phys. Rev. Lett. **113**, 230501 (2014).
[21] D. Poulin, Phys. Rev. Lett. **95**, 230504 (2005).
[22] D. Bacon, Phys. Rev. A **73**, 012340 (2006).
[23] P. Aliferis and A. W. Cross, Phys. Rev. Lett. **98**, 220502 (2007).
[24] H. Bombin, Phys. Rev. A **81**, 032301 (2010).
[25] H. Bombin, R. W. Chhajlany, M. Horodecki, and M. A. Martin-Delgado, New J. Phys. **15**, 055023 (2013).
[26] B. Eastin, Phys. Rev. A **87**, 032321 (2013).
[27] C. Jones, Phys. Rev. A **87**, 022328 (2013).
[28] R. Raussendorf, J. Harrington, and K. Goyal, New J. Phys. **9**, 199 (2007).
[29] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, New J. Phys. **14**, 123011 (2012).
[30] A. J. Landahl and C. Ryan-Anderson, arXiv:1407.5103.
[31] S. D. Barrett and P. Kok, Phys. Rev. A **71**, 060310 (2005).
[32] L. Jiang, J. M. Taylor, A. S. Sørensen, and M. D. Lukin, Phys. Rev. A **76**, 062323 (2007).
[33] D. K. L. Oi, S. J. Devitt, and L. C. L. Hollenberg, Phys. Rev. A **74**, 052313 (2006).
[34] D. L. Moehring, P. Maunz, S. Olmschenk, K. C. Younge, D. N. Matsukevich, L. M. Duan, and C. Monroe, Nature (London) **449**, 68 (2007).
[35] E. T. Campbell and S. C. Benjamin, Phys. Rev. Lett. **101**, 130502 (2008).
[36] E. Campbell and J. Fitzsimons, Int. J. Quantum Inf. **8**, 219 (2010).
[37] H. Bernien, B. Hensen, W. Pfaff, G. Koolstra, M. S. Blok, L. Robledo, T. H. Taminiau, M. Markham, D. J. Twitchen, L. Childress, and R. Hanson, Nature (London) **497**, 86 (2013).
[38] N. H. Nickerson, J. F. Fitzsimons, and S. C. Benjamin, Phys. Rev. X **4**, 041041 (2014).
[39] J. T. Anderson, arXiv:1205.0289.
[40] E. T. Campbell and M. Howard, Phys. Rev. Lett. **118**, 060501 (2017).
[41] E. T. Campbell and M. Howard, Phys. Rev. A **95**, 022316 (2017).
[42] Y. Li, New J. Phys. **17**, 023037 (2015).
[43] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Phys. Rev. A **86**, 032324 (2012).
[44] M. Reiher, N. Wiebe, K. M. Svore, D. Wecker, and M. Troyer, arXiv:1605.03590.
[45] A. Fowler (private communication).
[46] G. Duclos-Cianci and K. M. Svore, Phys. Rev. A **88**, 042325 (2013).
[47] A. J. Landahl and C. Cesare, arXiv:1302.3240.
[48] G. Duclos-Cianci and D. Poulin, Phys. Rev. A **91**, 042315 (2015).
[49] E. T. Campbell and J. O'Gorman, Quantum Sci. Technol. **1**, 015007 (2016).
[50] D. Herr, F. Nori, and S. J. Devitt, New J. Phys. **19**, 013034 (2017).
[51] See Supplemental Material at http://link.aps.org/supplemental/10.1103/PhysRevA.95.032338 for a Mathematica notebook which calculates the dual matrix of G(k) for the Bravyi-Haah codes and a list of the specific combinations of protocols which produced the data points in Fig. 5.